

INDEX

Ex. No.	Experiments	Page No.	Date of Experiment:	Signature
1	To implement addition and multiplication of two 2D arrays.			
2	To transpose a 2D array.			
3	To implement stack using array.			
4	To implement queue using array.			
5	To implement circular queue using array.			
6	To implement stack using linked list.			
7	To implement queue using linked list.			
8	To implement BFS using linked list.			
9	To implement DFS using linked list.			
10	To implement Linear Search.			
11	To implement Binary Search.			
12	To implement Bubble Sorting.			
13	To implement Selection Sorting.			
14	To implement Insertion Sorting.			
15	To implement Merge Sorting.			
16	To implement Heap Sorting.			
17	To implement Matrix Multiplication by strassen's algorithm.			
18	Find Minimum Spanning Tree using Kruskal's Algorithm.			

Experiment No: 1

Problem Statement: Write a program to add and multiply 2D arrays.

1 (a). Write a programme for addition of 2D arrays.

```
// Addition of 2D array
#include <stdio.h>
#include <conio.h>
#define N 50
int main() {
    int a[N][N], b[N][N], c[N][N] = {0};
    int i, j, m, n, p, q;
    printf("Enter no. of rows and columns in matrix A: \n");
    scanf("%d%d", &m, &n);
    printf("Enter no. of rows and columns in matrix B: \n");
    scanf("%d%d", &p, &q);
    if (m != p || n != q) {
        printf("2D array Addition is not possible");
        return 0 ;
    } else {
        printf("Enter elements of 2D array A: \n");
        for (i = 0; i < m; i++)
            for (j = 0; j < n; j++)
                scanf("%d", &a[i][j]);

        printf("Enter elements of 2D Array B: \n");
        for (i = 0; i < p; i++)
            for (j = 0; j < q; j++)
                scanf("%d", &b[i][j]);

        // Matrix Addition
```

```

    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            c[i][j] = a[i][j] + b[i][j];

printf("\nResult of 2D array Addition:\n");
for (i = 0; i < m; i++){
    for (j = 0; j < n; j++){
        printf("%d ", c[i][j]);
    }
    printf("\n");
}

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>

```

1 (b). Write a programme for multiplication of 2D arrays.

```
#include <stdio.h>
#include <conio.h>
#define N 50
int main(){
    int a[N][N], b[N][N], c[N][N] = {0};
    int i, j, k, m, n, p, q;
    printf("Enter no. of rows and columns in matrix A:");
    scanf("%d%d", &m, &n);
    printf("Enter no. of rows and columns in matrix B:");
    scanf("%d%d", &p, &q);
    if (n != p){
        printf("2D array Multiplication is not
possible");
        return;
    }else{
        printf("Enter elements of 2D array A: \n");
        for (i = 0; i < m; i++)
            for (j = 0; j < n; j++)
                scanf("%d", &a[i][j]);
        printf("Enter elements of 2D Array B: \n");
        for (i = 0; i < p; i++)
            for (j = 0; j < q; j++)
                scanf("%d", &b[i][j]);
        // Array Multiplication
        for (i = 0; i < m; i++)
            for (j = 0; j < q; j++)
                for (k = 0; k < p; k++)
                    c[i][j] += a[i][k] * b[k][j];
        printf("\nResult of 2D array Multiplication:\n");
        for (i = 0; i < m; i++){
            for(j=0;j<q;j++){
```

```

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

    }

    printf("\n");

}

}

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> .\'arrayMul.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.

```
#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;
}
```

```
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> |
```

OUTPUT : -

Experiment No: 3

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

```
#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\out
put'
• 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.

```

// Q4. To implement queue using array.

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

#define MAX_SIZE 5

// Define a structure for the queue
struct Queue {
    int arr[MAX_SIZE];
    int front, rear;
};

// Function to initialize a queue
void initQueue(struct Queue* queue) {
    queue->front = -1;
    queue->rear = -1;
}

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

```

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

// Function to add an element to the queue
void enqueue(struct Queue* queue, int data) {
    if (isFull(queue)) {
        printf("Queue is full. Cannot enqueue.\n");
        return;
    }
    if (isEmpty(queue)) {
        queue->front = 0;
        queue->rear = 0;
    } else {
        queue->rear = (queue->rear + 1) % MAX_SIZE;
    }
    queue->arr[queue->rear] = data;
    printf("%d enqueued to queue.\n", data);
}

// Function to remove an element from the queue
int dequeue(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    }
    int removedElement = queue->arr[queue->front];
    if (queue->front == queue->rear) {
        queue->front = -1;
        queue->rear = -1;
    } else {
```

```
        queue->front = (queue->front + 1) % MAX_SIZE;
    }
    return removedElement;
}

// Function to get the front element of the queue
int front(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return -1;
    }
    return queue->arr[queue->front];
}

// Function to get the rear element of the queue
int rear(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return -1;
    }
    return queue->arr[queue->rear];
}

int main() {
    struct Queue queue;
    initQueue(&queue);

    enqueue(&queue, 10);
    enqueue(&queue, 20);
    enqueue(&queue, 30);

    printf("Front element is %d.\n", front(&queue));
}
```

```

printf("Rear element is %d.\n", rear(&queue));

printf("%d dequeued from queue.\n", dequeue(&queue));
printf("%d dequeued from queue.\n", dequeue(&queue));

printf("Front element is %d.\n", front(&queue));
printf("Rear element is %d.\n", rear(&queue));

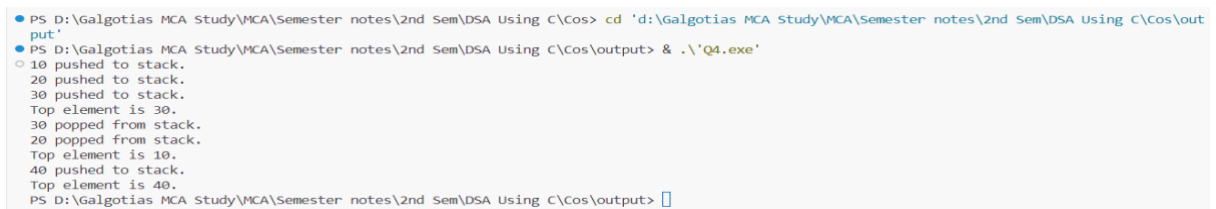
enqueue(&queue, 40);
enqueue(&queue, 50);
enqueue(&queue, 60);
enqueue(&queue, 70);
enqueue(&queue, 80); // Queue is full here

printf("Front element is %d.\n", front(&queue));
printf("Rear element is %d.\n", rear(&queue));

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\out
put'
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.

```
// circular queue using array
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 5
// Define a structure for the circular queue
struct CircularQueue {
    int arr[MAX_SIZE];
    int front, rear;
};
// Function to initialize a circular queue
void initCircularQueue(struct CircularQueue* queue) {
    queue->front = -1;
    queue->rear = -1;
}
// Function to check if the circular queue is empty
int isCircularQueueEmpty(struct CircularQueue* queue) {
    return (queue->front == -1);
}
// Function to check if the circular queue is full
int isCircularQueueFull(struct CircularQueue* queue) {
    return ((queue->rear + 1) % MAX_SIZE == queue->front);
}
// Function to add an element to the circular queue
```

```
void enqueue(struct CircularQueue* queue, int data) {
    if (isCircularQueueFull(queue)) {
        printf("Circular queue is full. Cannot enqueue.\n");
        return;
    }
    if (isCircularQueueEmpty(queue)) {
        queue->front = 0;
    }
    queue->rear = (queue->rear + 1) % MAX_SIZE;
    queue->arr[queue->rear] = data;
    printf("%d enqueued to circular queue.\n", data);
}

// Function to remove an element from the circular queue
int dequeue(struct CircularQueue* queue) {
    if (isCircularQueueEmpty(queue)) {
        printf("Circular queue is empty. Cannot dequeue.\n");
        return -1;
    }
    int removedElement = queue->arr[queue->front];
    if (queue->front == queue->rear) {
        queue->front = -1;
        queue->rear = -1;
    } else {
        queue->front = (queue->front + 1) % MAX_SIZE;
    }
    return removedElement;
}

// Function to get the front element of the circular queue
int front(struct CircularQueue* queue) {
    if (isCircularQueueEmpty(queue)) {
        printf("Circular queue is empty.\n");
        return -1;
    }
}
```

```
    }  
    return queue->arr[queue->front];  
}  
  
// Function to get the rear element of the circular queue  
int rear(struct CircularQueue* queue) {  
    if (isCircularQueueEmpty(queue)) {  
        printf("Circular queue is empty.\n");  
        return -1;  
    }  
    return queue->arr[queue->rear];  
}  
  
int main() {  
    struct CircularQueue queue;  
    initCircularQueue(&queue);  
    enqueue(&queue, 10);  
    enqueue(&queue, 20);  
    enqueue(&queue, 30);  
    printf("Front element is %d.\n", front(&queue));  
    printf("Rear element is %d.\n", rear(&queue));  
    printf("%d dequeued from circular queue.\n",  
    dequeue(&queue));  
    printf("%d dequeued from circular queue.\n",  
    dequeue(&queue));  
    printf("Front element is %d.\n", front(&queue));  
    printf("Rear element is %d.\n", rear(&queue));  
    enqueue(&queue, 40);  
    enqueue(&queue, 50);  
    enqueue(&queue, 60);  
    enqueue(&queue, 70);  
    enqueue(&queue, 80); // Circular queue is full here  
    printf("Front element is %d.\n", front(&queue));  
    printf("Rear element is %d.\n", rear(&queue));  
    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\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> █
```