

Experiment No. 1**Aim:**

Write a program to Add two matrices

Theory:

1. Input: The program asks the user to enter the number of rows and columns for the matrices, followed by the elements for both matrices.
2. Matrix Addition: The add_matrices function adds corresponding elements of the two matrices and stores the result in the result matrix.
3. Output: The display_matrix function is used to print the resultant matrix.

Program:

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

```
#define MAX 10
```

```
void add_matrices(int matrix1[][MAX], int matrix2[][MAX], int result[][MAX], int rows, int cols) {  
    for (int i = 0; i < rows; i++) {  
        for (int j = 0; j < cols; j++) {  
            result[i][j] = matrix1[i][j] + matrix2[i][j];  
        }  
    }  
}
```

```
void display_matrix(int matrix[][MAX], int rows, int cols) {  
    for (int i = 0; i < rows; i++) {  
        for (int j = 0; j < cols; j++) {  
            printf("%d ", matrix[i][j]);  
        }  
        printf("\n");  
    }  
}
```

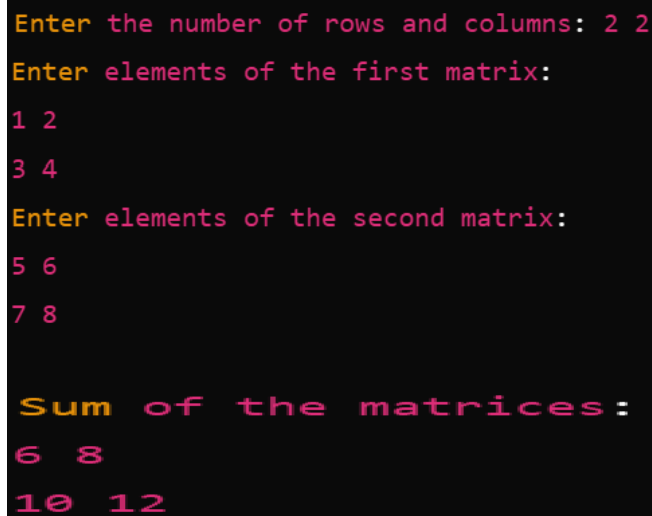
```
int main() {  
    int matrix1[MAX][MAX], matrix2[MAX][MAX], result[MAX][MAX];  
    int rows, cols;  
  
    printf("Enter the number of rows and columns: ");  
    scanf("%d %d", &rows, &cols);  
  
    printf("Enter elements of the first matrix:\n");  
    for (int i = 0; i < rows; i++) {  
        for (int j = 0; j < cols; j++) {  
            scanf("%d", &matrix1[i][j]);  
        }  
    }  
}
```

```
printf("Enter elements of the second matrix:\n");
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        scanf("%d", &matrix2[i][j]);
    }
}

add_matrices(matrix1, matrix2, result, rows, cols);

printf("Sum of the matrices:\n");
display_matrix(result, rows, cols);

return 0;
}
```

Output:

```
Enter the number of rows and columns: 2 2
Enter elements of the first matrix:
1 2
3 4
Enter elements of the second matrix:
5 6
7 8

Sum of the matrices:
6 8
10 12
```

Conclusion:

The program checks that the matrices are of compatible sizes before performing the addition.

It uses simple loops to iterate through the elements of both matrices and adds them.

The result is displayed in a clear, matrix-like format.

By understanding this program, one can apply similar principles to perform other matrix operations such as multiplication, subtraction, or even transpose. Matrix operations form the foundation for more complex numerical methods and are widely used in areas like computer graphics, machine learning, and scientific computing.

Experiment No. 2**Aim:**

Write a Program to find the sum of the elements of the upper triangle of a Square matrix

Theory:

1. Matrix Input: The program first asks the user to input the size of the square matrix (n x n) and then the elements of the matrix.
2. Upper Triangle Sum: The `sum_upper_triangle` function calculates the sum of the elements of the upper triangle of the matrix. The upper triangle consists of elements where the row index is less than or equal to the column index ($i \leq j$).
3. Output: The program then prints the sum of the upper triangle elements.

Program:

```
#include <stdio.h>
#define MAX 10
int sum_upper_triangle(int matrix[][MAX], int n) {
    int sum = 0;

    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            sum += matrix[i][j];
        }
    }
    return sum;
}
int main() {
    int matrix[MAX][MAX];
    int n;

    printf("Enter the size of the square matrix (n x n): ");
    scanf("%d", &n);

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

    int result = sum_upper_triangle(matrix, n);

    printf("Sum of the elements of the upper triangle: %d\n", result);

    return 0;
}
```

Output:

```
Enter the size of the square matrix (n x n): 3
Enter the elements of the matrix:
1 2 3
4 5 6
7 8 9
```

```
Sum of the elements of the upper triangle: 26
```

Conclusion:

This program effectively calculates the sum of the elements in the upper triangle of a square matrix, providing a simple yet useful operation for matrix-related tasks. This can be extended to other matrix properties like lower triangle sums or even more complex calculations in linear algebra and scientific computing.

Experiment No. 3**Aim:**

Write a Program to find the sum of the elements of the Lower triangle of a Square matrix

Theory:

1. Matrix Input: The program first asks the user to input the size of the square matrix (n x n) and the elements of the matrix.
2. Lower Triangle Sum: The `sum_lower_triangle` function calculates the sum of the elements in the lower triangle of the matrix. The lower triangle consists of elements where the row index is greater than or equal to the column index ($i \geq j$).
3. Output: The program then prints the sum of the lower triangle elements

Program:

```
#include <stdio.h>
#define MAX 10

int sum_lower_triangle(int matrix[][MAX], int n) {
    int sum = 0;

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

    return sum;
}

int main() {
    int matrix[MAX][MAX];
    int n;

    printf("Enter the size of the square matrix (n x n): ");
    scanf("%d", &n);

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

    int result = sum_lower_triangle(matrix, n);
    printf("Sum of the elements of the lower triangle: %d\n", result); return 0;
}
```

Output

:

```
Enter the size of the square matrix (n x n): 3
Enter the elements of the matrix:
1 2 3
4 5 6
7 8 9
Sum of the elements of the lower triangle: 28
```

Conclusion:

This program successfully calculates the sum of the elements in the lower triangle of a square matrix. The lower triangle of a matrix includes all elements where the row index is greater than or equal to the column index. This operation is helpful in various matrix computations, especially in solving problems related to triangular matrices in linear algebra.

Experiment No. 4**Aim:**

Write a Program to find the sum of the elements of the Upper & Lower triangle of a Square matrix

Theory:

1. Matrix Input: The program prompts the user to enter the size of the square matrix (n x n) and the elements of the matrix.
2. Upper Triangle Sum: The `sum_upper_triangle` function calculates the sum of the elements in the upper triangle, including the diagonal. It iterates over the matrix, considering elements where $i \leq j$.
3. Lower Triangle Sum: The `sum_lower_triangle` function calculates the sum of the elements in the lower triangle, including the diagonal. It iterates over the matrix, considering elements where $i \geq j$.
4. Output: The program displays the sum of both the upper and lower triangle elements.

Program:

```
#include<stdio.h>

#define MAX 10

int sum_upper_triangle(int matrix[][MAX], int n) {
    int sum = 0;

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

    return sum;
}

int sum_lower_triangle(int matrix[][MAX], int n) {
    int sum = 0;

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

```
    return sum;
}

int main() {
    int matrix[MAX][MAX];
    int n;

    printf("Enter the size of the square matrix (n x n): ");
    scanf("%d", &n);

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

    int upper_sum = sum_upper_triangle(matrix, n);
    int lower_sum = sum_lower_triangle(matrix, n);

    // Output the results
    printf("Sum of the elements of the upper triangle: %d\n", upper_sum); printf("Sum
of the elements of the lower triangle: %d\n", lower_sum);

    return 0;
}
```

Output:

```
Enter the size of the square matrix (n x n): 3
Enter the elements of the matrix:
1 2 3
4 5 6
7 8 9

Sum of the elements of the upper triangle: 26
Sum of the elements of the lower triangle: 28
```

Conclusion:

This program efficiently calculates the sum of the elements in both the upper and lower triangles of a square matrix. The upper triangle includes elements where the row index is less than or equal to the column index, while the lower triangle includes elements where the row index is greater than or equal to the column index. This can be useful in matrix operations, particularly in problems dealing with triangular matrices in numerical analysis.

Experiment No. 5**Aim:**

Write a Program to transpose a given square matrix

Theory:

1. **Matrix Input:** The program first asks the user to input the size of the square matrix ($n \times n$) and the elements of the matrix.
2. **Transpose Operation:** The `transpose_matrix` function creates the transpose of the matrix. It swaps the rows and columns, i.e., the element at position (i, j) in the original matrix is moved to position (j, i) in the transposed matrix.
3. **Matrix Display:** The `display_matrix` function is used to print both the original and the transposed matrices.
4. **Output:** The program displays the original matrix and its transposed version.

Program:

```
#include <stdio.h>

#define MAX 10

void transpose_matrix(int matrix[][MAX], int transpose[][MAX], int n) {
    // Transpose the matrix
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            transpose[i][j] = matrix[j][i];
        }
    }
}

void display_matrix(int matrix[][MAX], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int matrix[MAX][MAX], transpose[MAX][MAX];
    int n;
    printf("Enter the size of the square matrix (n x n): ");
    scanf("%d", &n);
    printf("Enter the elements of the matrix:\n");
    for (int i = 0; i < n; i++) {
```

```
        for (int j = 0; j < n; j++) {  
            scanf("%d", &matrix[i][j]);  
        }  
    }  
    transpose_matrix(matrix, transpose, n);  
  
    printf("\nOriginal Matrix:\n");  
    display_matrix(matrix, n);  
  
    printf("\nTransposed Matrix:\n");  
    display_matrix(transpose, n);  
  
    return 0;  
}
```

Output:

```
Enter the size of the square matrix (n x n): 3  
Enter the elements of the matrix:  
1 2 3  
4 5 6  
7 8 9  
  
Original Matrix:  
1 2 3  
4 5 6  
7 8 9  
  
Transposed Matrix:  
1 4 7  
2 5 8  
3 6 9
```

Conclusion:

This program successfully transposes a square matrix by swapping its rows and columns. The transpose operation is commonly used in various matrix-related computations, such as in solving linear equations, computer graphics, and other mathematical application

Experiment No. 6**Aim:**

Write a Program to multiply the two matrices

Theory:

1. Matrix Input: The program first asks the user to input the dimensions and elements of both matrices.
2. Matrix Multiplication: The multiply_matrices function performs the matrix multiplication. It checks if the number of columns in the first matrix is equal to the number of rows in the second matrix (which is required for multiplication). Then, it calculates the product of the two matrices and stores the result in a third matrix.
3. Matrix Display: The display_matrix function is used to print the resulting matrix.
4. Output: The program displays the product of the two matrices.

Program:

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

```
#define MAX 10
```

```
void multiply_matrices(int matrix1[][MAX], int matrix2[][MAX], int result[][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 i = 0; i < row1; i++) {  
        for (int j = 0; j < col2; j++) {  
            for (int k = 0; k < col1; k++) {  
                result[i][j] += matrix1[i][k] * matrix2[k][j];  
            }  
        }  
    }
```

```
}
```

```
void display_matrix(int matrix[][MAX], int row, int col) {  
    for (int i = 0; i < row; i++) {  
        for (int j = 0; j < col; j++) {
```

```
        printf("%d ", matrix[i][j]);
    }
    printf("\n");
}

int main() {
    int matrix1[MAX][MAX], matrix2[MAX][MAX], result[MAX][MAX];
    int row1, col1, row2, col2;

    printf("Enter the number of rows and columns for matrix 1: ");
    scanf("%d %d", &row1, &col1);
    printf("Enter the elements of the first matrix:\n");
    for (int i = 0; i < row1; i++) {
        for (int j = 0; j < col1; j++) {
            scanf("%d", &matrix1[i][j]);
        }
    }
    printf("Enter the number of rows and columns for matrix 2: ");
    scanf("%d %d", &row2, &col2);
    printf("Enter the elements of the second matrix:\n");
    for (int i = 0; i < row2; i++) {
        for (int j = 0; j < col2; j++) {
            scanf("%d", &matrix2[i][j]);
        }
    }
    multiply_matrices(matrix1, matrix2, result, row1, col1, row2, col2);

    printf("\nProduct of the matrices:\n");
    display_matrix(result, row1, col2);

    return 0;
}
```

Output:

```
Enter the number of rows and columns for matrix 1: 2 3
Enter the elements of the first matrix:
1 2 3
4 5 6
Enter the number of rows and columns for matrix 2: 3 2
Enter the elements of the second matrix:
7 8
9 10
11 12

Product of the matrices:
58 64
139 154
```

Conclusion:

This program efficiently multiplies two matrices if their dimensions are compatible. Matrix multiplication is fundamental in many fields, such as computer graphics, physics simulations, machine learning, and more. The program checks for dimension compatibility before performing the multiplication and outputs the resulting matrix.