

Experiment No. 1

Aim: Write a program to evaluate C using function where $C=A+B$ with return

Theory:

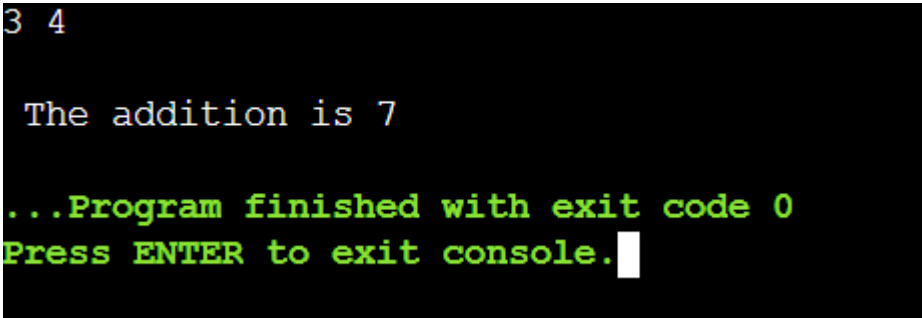
Here are the important theoretical points for your code:

1. **Modular Programming:** The `add()` function demonstrates the modular approach by separating the addition logic, improving readability and reusability.
2. **Function Usage:** It showcases the use of a user-defined function that returns a value to the calling function.
3. **Standard I/O Functions:** Uses `scanf()` for input and `printf()` for output, part of the standard C library.
4. **Basic Arithmetic:** Performs addition using variables and the `+` operator.
5. **Control Flow:** Demonstrates function calls and the return mechanism in C.

Code:

```
#include <stdio.h> int add(int a, int b){  
  
int c; c= a+b;  
  
return c;  
  
}  
  
int main() { int a,b,c;  
  
scanf("%d%d",&a,&b); c= add(a,b);  
  
printf("\n The addition is %d",c);  
  
return 0;  
  
}
```

Output:



```
3 4  
  
The addition is 7  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Conclusion:

The C program demonstrates the use of a user-defined function, `add()`, to compute the sum of two integers. It takes user input via `scanf()`, passes the values to the function, and outputs the result using `printf()`. This modular approach enhances code clarity and reusability. A minor syntax correction is needed in `return 0;`. Overall, it's a basic yet effective example of functions, input/output, and structured programming for beginners.

Experiment No 2:

Aim: Write a program to evaluate C using function where $C=A+B$ without return

Theory:

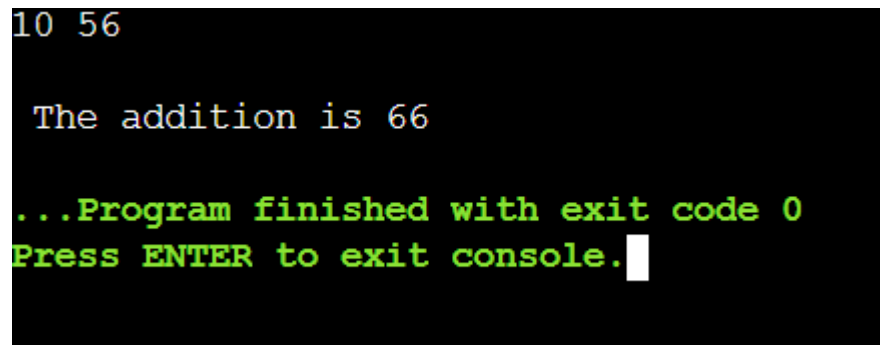
Here are the key theoretical points based on the code:

- 1.Function Without Return: The add() function demonstrates a void function, which performs a task (addition and printing) without returning a value.
- 2.Parameter Passing: The function uses call by value, where the values of a and b are passed as arguments.
- 3.Standard Input/Output: Utilizes scanf() for input and printf() for displaying the result.

Code:

```
#include<stdio.h> void add(int a, int b){  
  
int c; c= a+b;  
  
printf("\n The addition is %d",c);  
  
}  
  
int main() { int a,b,c;  
  
scanf("%d%d",&a,&b); add(a,b);  
  
return 0;  
  
}
```

Output:



```
10 56  
  
The addition is 66  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Conclusion:

This C program demonstrates the use of a 'void' function, 'add()', to compute and directly display the sum of two integers. The 'add()' function takes two inputs, calculates their sum, and prints the result using 'printf()'. User input is taken via 'scanf()' in the 'main()' function, which then calls the 'add()' function. This approach simplifies the code by performing calculation and output in the same function, making it suitable for basic programming practice.

Experiment No 3

Aim: Write a program to find the area of a Triangle using Functions $\text{Area} = \sqrt{S(S-a)(S-b)(S-c)}$ where $S = (a+b+c)/2.0$

Theory:

Here are the key theoretical points for the triangle area program:

- 1.Modular Programming: The calculateArea function encapsulates the logic for calculating the triangle's area, enhancing readability and reusability.
- 2.Math Library Usage: The program uses the sqrt() function from the <math.h> library to compute the square root.

Code:

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

float calculateArea(float a, float b, float c) { float s, area;

s = (a + b + c) / 2.0; // Semi-perimeter area = sqrt(s * (s - a) * (s - b) * (s - c)); return area;

}

int main() {

float a, b, c, area;

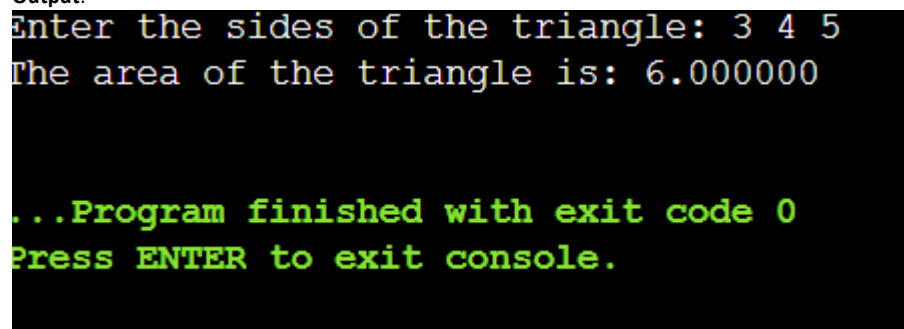
printf("Enter the sides of the triangle: "); scanf("%f %f %f", &a, &b, &c);

area = calculateArea(a, b, c);

printf("The area of the triangle is: %f\n", area); return 0;

}
```

Output:



```
Enter the sides of the triangle: 3 4 5
The area of the triangle is: 6.000000
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:

This C program calculates the area of a triangle using Heron's formula. It features a user-defined function, 'calculateArea()', which computes the semi-perimeter and uses it to determine the area. The program takes the triangle's side lengths as input via 'scanf()', computes the area using the 'sqrt()' function, and displays the result with 'printf()'. It effectively demonstrates the use of mathematical formulas, modular programming, and function design, making it a practical example for learning basic geometry in C.

Experiment No 4

Aim: Write a program to find NCR where $NCR = N! / R!(N-R)!$

Theory:

1. Factorial Calculation: The factorial function computes the factorial of a number iteratively.
2. Combination Formula: The program implements the formula $NCR = N! / (R!(N-R)!)$
3. Validation: Ensures $R \leq N$ and $N \geq 0$ and both values are non-negative.

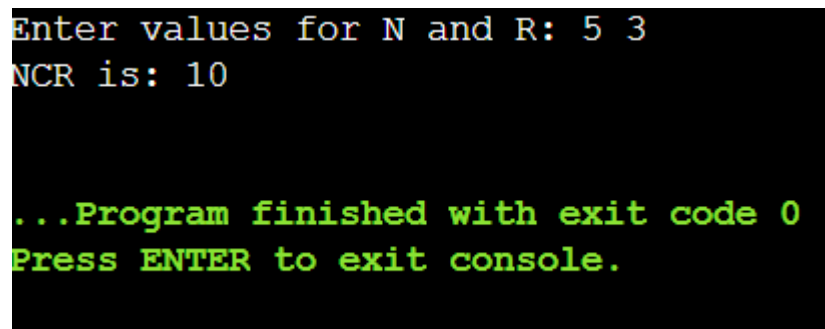
Code:

```
#include <stdio.h>

long long factorial(int num) { long long fact = 1;
    for (int i = 1; i <= num; i++) { fact *= i;
    }
    return fact;
}

int main() { int n, r;
    long long ncr;
    printf("Enter values for N and R: "); scanf("%d %d", &n, &r);
    ncr = factorial(n) / (factorial(r) * factorial(n - r)); printf("NCR is: %lld\n", ncr);
    return 0;
}
```

Output:



```
Enter values for N and R: 5 3
NCR is: 10

...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:

This C program calculates the combination (NCR) using the formula $n! / r!(n-r)!$. It defines a

`factorial()` function to compute the factorial of a given number using a loop. In the

`main()` function, it takes two integer inputs, N and R, and computes NCR by calling the

`factorial()` function. The result is then printed using `printf()`. This program efficiently demonstrates basic factorial calculation and combination formula implementation in C.

Experiment No 5

Aim: Write a program using function to evaluate the following expression $T = \sum_{i=1}^n (X_i * Y_i * Z_i) / K_1! + \sum_{i=1}^n (X_i^2 Y_i) / K_2!$

$+ \sum_{i=1}^n (X_i Y_i^2) / K_3!$

Theory:

1. Factorial Calculation: The factorial() function computes the factorial of a number iteratively.

2. Series Calculation:

calculateTerm1 computes the first summation term. calculateTerm2 computes the second summation term. calculateTerm3 computes the third summation term.

3. Inputs: The program reads n, arrays X,Y,Z and constants K1,K2,K3 .

4. Summation: The program adds up the results of the three terms to calculate T.

Code:

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

long long factorial(int k) { long long fact = 1;

for (int i = 1; i <= k; i++) { fact *= i;

}

return fact;

}

double calculateTerm1(int n, double x[], double y[], double z[], int k1) { double sum = 0;

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

sum += (x[i] * y[i] * z[i]) / factorial(k1);

}

return sum;

}

double calculateTerm2(int n, double x[], double y[], int k2) { double sum = 0;

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

sum += (pow(x[i], 2) * y[i]) / factorial(k2);

}

return sum;

}

double calculateTerm3(int n, double x[], double y[], int k3) { double sum = 0;

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

sum += (x[i] * pow(y[i], 2)) / factorial(k3);

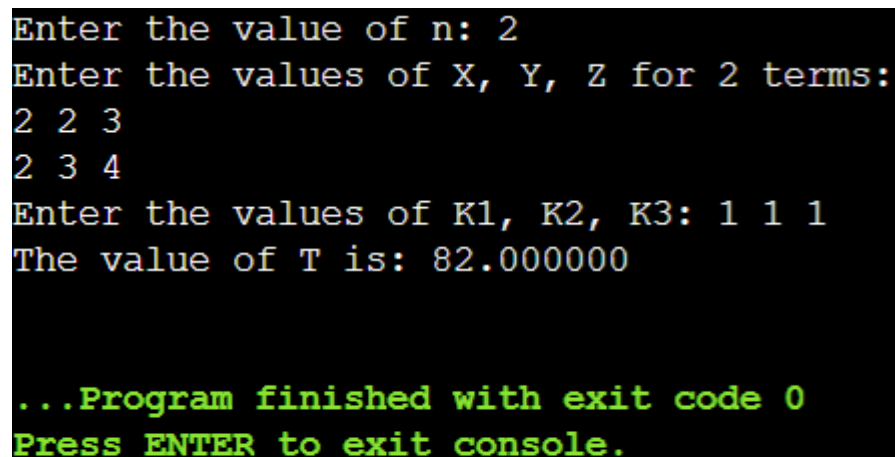
}

return sum;

}
```

```
int main() {  
  
    int n, k1, k2, k3;  
  
    printf("Enter the value of n: "); scanf("%d", &n);  
  
    double x[n], y[n], z[n];  
  
    printf("Enter the values of X, Y, Z for %d terms:\n", n); for (int i = 0; i < n; i++) {  
  
        scanf("%lf %lf %lf", &x[i], &y[i], &z[i]);  
  
    }  
  
    printf("Enter the values of K1, K2, K3: "); scanf("%d %d %d", &k1, &k2, &k3);  
  
    double T = calculateTerm1(n, x, y, z, k1) + calculateTerm2(n, x, y, k2) + calculateTerm3(n, x, y, k3);  
  
    printf("The value of T is: %f\n", T); return 0;  
  
}
```

Output:



```
Enter the value of n: 2  
Enter the values of X, Y, Z for 2 terms:  
2 2 3  
2 3 4  
Enter the values of K1, K2, K3: 1 1 1  
The value of T is: 82.000000  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Conclusion:

This C program calculates the value of T by evaluating three separate terms using user input values for arrays x, y, and z, as well as constants k1, k2, and k3. It defines three functions,

`calculateTerm1()`, `calculateTerm2()`, and `calculateTerm3()`, each computing a specific term involving factorials and mathematical operations. The results are summed to compute T, demonstrating advanced calculations with arrays, loops, and factorial functions in C.

Experiment No. 6

Aim: Write a program to exemplify method reference by another method $T = \text{Summation } 1 \text{ to } L(X_i)^2 / \text{fact}(K) + \text{Summation } 1 \text{ to } L(X_i)^2 / \text{fact}(N)$

Theory:

1. Factorial Calculation: The factorial function computes the factorial of a number.
2. Function Pointer: The long long (*factMethod)(int) parameter in calculateSummation allows method reference-like functionality, enabling dynamic function calls.
3. Summation Calculation: The program calculates summation terms by iterating over array values and dividing by the factorial of constants K and N.

Code:

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

long long factorial(int num) { long long fact = 1;

for (int i = 1; i <= num; i++) { fact *= i;

}

return fact;

}

double calculateSummation(int x[], int length, int constant, long long (*factMethod)(int)) { double sum = 0;

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

sum += pow(x[i], 2) / factMethod(constant);

}

return sum;

}

int main() { int L, k, n;

printf("Enter the number of terms (L): "); scanf("%d", &L);

int x[L];

printf("Enter the values of X (separated by spaces): "); for (int i = 0; i < L; i++) {

scanf("%d", &x[i]);

}

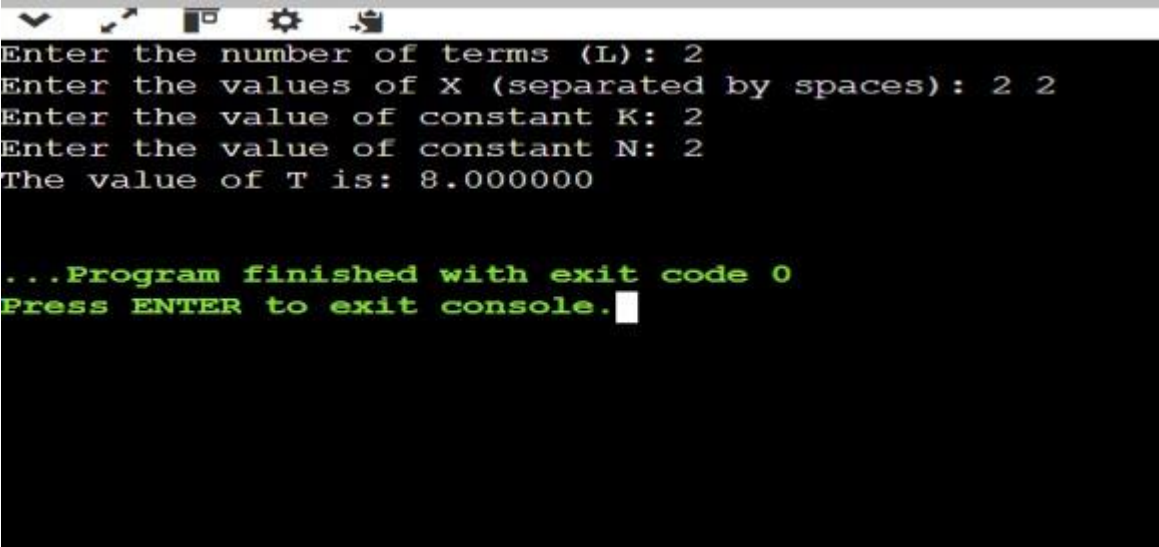
printf("Enter the value of constant K: "); scanf("%d", &k);

printf("Enter the value of constant N: "); scanf("%d", &n);

double term1 = calculateSummation(x, L, k, factorial); double term2 = calculateSummation(x, L, n, factorial); double T = term1 + term2;

printf("The value of T is: %f\n", T); return 0;

}
```

Output:A screenshot of a terminal window with a black background and white text. The window has a standard OS title bar at the top with icons for minimize, maximize, and close. The text in the terminal shows a C program running. It prompts the user for the number of terms (L), the values of X (separated by spaces), the value of constant K, and the value of constant N. The user has entered 2 for L, 2 2 for X, 2 for K, and 2 for N. The program then outputs 'The value of T is: 8.000000'. Below this, it says '...Program finished with exit code 0' and 'Press ENTER to exit console.' with a white cursor box at the end of the line.

```
Enter the number of terms (L): 2
Enter the values of X (separated by spaces): 2 2
Enter the value of constant K: 2
Enter the value of constant N: 2
The value of T is: 8.000000

...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:

The program efficiently calculates the value of T by summing two terms that involve array elements and factorial calculations. It demonstrates function usage in C, employing functions like `factorial()` for computing factorial values and `calculateSummation()` to process the terms based on user inputs. The program also makes use of arrays and pointers for handling data input, processing, and output, providing a clear example of structured programming. This showcases how C can be used for mathematical computations and the handling of user-defined functions.

Experiment No. 7

Aim: Write a programme using functions to verify $([A] * [B])^T = [B]^T * [A]^T$

Theory:

1. Matrix Multiplication: The function multiplyMatrices multiplies two matrices.
2. Transpose: The transposeMatrix function transposes a matrix.
3. Verification: The program computes both $(A \times B)^T$ and $B^T \times A^T$ and compares the results. The matrices are input by the user, and the program displays both results to verify the identity.

Code:

```
#include <stdio.h>

void multiplyMatrices(int A[100][100], int B[100][100], int result[100][100], int rowA, int colA, int rowB, int colB) {
    for (int i = 0; i < rowA; i++) { for (int j = 0; j < colB; j++) {
        result[i][j] = 0;
        for (int k = 0; k < colA; k++) { result[i][j] += A[i][k] * B[k][j];
        }
    }
}

void transposeMatrix(int matrix[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[j][i] = matrix[i][j];
    }
}

void displayMatrix(int matrix[MAX][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 A[100][100], B[100][100], AB[100][100], AT[100][100], BT[100][100], BT_A_T[100][100], ABT[100][100];
    int rowA, colA, rowB, colB;

    printf("Enter number of rows and columns for matrix A: "); scanf("%d %d", &rowA, &colA);

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

```

}

printf("Enter number of rows and columns for matrix B: ");

scanf("%d %d", &rowB, &colB); if (colA != rowB) {

printf("Matrix multiplication is not possible. Number of columns of A must equal the number of rows of B.\n");

return 1;

}

printf("Enter elements of matrix B:\n"); for (int i = 0; i < rowB; i++) {

for (int j = 0; j < colB; j++) { scanf("%d", &B[i][j]);

}

}

multiplyMatrices(A, B, AB, rowA, colA, rowB, colB);

transposeMatrix(A, AT, rowA, colA); transposeMatrix(B, BT, rowB, colB); transposeMatrix(AB, ABT, rowA, colB);

multiplyMatrices(BT, AT, BT_A_T, colB, rowB, colA, rowA); printf("Matrix (A * B)^T:\n");

displayMatrix(ABT, rowA, colB); printf("Matrix B^T * A^T:\n"); displayMatrix(BT_A_T, colB, rowA);

return 0;

}

```

Output:

```

Enter number of rows and columns for matrix A: 2 2
Enter elements of matrix A:
1 1 2 2
Enter number of rows and columns for matrix B: 2 2
Enter elements of matrix B:
2 2 1 2
Matrix (A * B)^T:
3 6
4 8
Matrix B^T * A^T:
3 6
4 8

=== Code Execution Successful ===

```

Conclusion:

This C program demonstrates matrix operations, including multiplication, transposition, and comparing two results. It takes two matrices, A and B, as input from the user, checks if multiplication is feasible, and computes the product $A \times B$. The program then calculates and displays the transpose of $A \times B$ and compares it with the product of the transposes of B and A. It efficiently illustrates matrix manipulation, handling multidimensional arrays and using multiple functions for each operation.

Experiment No 8

Aim: Write a program using functions to evaluate $T = [A] * [B] + [([C] * [D])^2] T + [A]^4$

Theory:

1. **Matrix Multiplication:** This code handles matrix multiplication between matrices $A \times B$ & $C \times D$ using the standard multiplication method.
2. **Matrix Transposition:** The result of $(C \times D)$ is transposed.
3. **Power of Matrix:** Each element of matrix A is raised to the power of 4 by repeated multiplication
4. **Matrix Addition:** Element-wise addition of matrices is performed to compute the final result T .

Code:

```
#include <stdio.h> #include <math.h> #define MAX=100

void multiplyMatrices(int A[MAX][MAX], int B[MAX][MAX], int result[MAX][MAX], int rowA, int colA, int rowB, int colB) {

    for (int i = 0; i < rowA; i++) { for (int j = 0; j < colB; j++) {

        result[i][j] = 0;

        for (int k = 0; k < colA; k++) { result[i][j] += A[i][k] * B[k][j];

        }

    }

}

void transposeMatrix(int matrix[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[j][i] = matrix[i][j];

    }

}

void powerMatrix(int matrix[MAX][MAX], int result[MAX][MAX], int row, int col, int power) { for (int i = 0; i < row; i++) {

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

        result[i][j] = pow(matrix[i][j], power);

    }

}

void displayMatrix(int matrix[MAX][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 A[MAX][MAX], B[MAX][MAX], C[MAX][MAX], D[MAX][MAX], AB[MAX][MAX],
CD[MAX][MAX], CD_squared[MAX][MAX], AT[MAX][MAX], A4[MAX][MAX];

int rowA, colA, rowB, colB, rowC, colC, rowD, colD; printf("Enter number of rows and columns for matrix A: "); scanf("%d %d",
&rowA, &colA);

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

printf("Enter number of rows and columns for matrix B: "); scanf("%d %d", &rowB, &colB);

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

printf("Enter number of rows and columns for matrix C: "); scanf("%d %d", &rowC, &colC);

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

printf("Enter number of rows and columns for matrix D: "); scanf("%d %d", &rowD, &colD);

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

multiplyMatrices(A, B, AB, rowA, colA, rowB, colB);

multiplyMatrices(C, D, CD, rowC, colC, rowD, colD);

multiplyMatrices(CD, CD, CD_squared, rowC, colD, rowC, colD);

transposeMatrix(CD_squared, AT, rowC, colD);

multiplyMatrices(A, A, AB, rowA, colA, rowA, colA); multiplyMatrices(AB, AB, A4, rowA, colA, rowA, colA);

for (int i = 0; i < rowA; i++) { for (int j = 0; j < colB; j++) {
    AB[i][j] += AT[i][j] + A4[i][j];
}
}

printf("Resulting matrix T:\n"); displayMatrix(AB, rowA, colB);

return 0;
}

```

Output:

Output

```
Enter number of rows and columns for matrix A: 2 2
Enter elements of matrix A:
1 1 1 1
Enter number of rows and columns for matrix B: 2 2
Enter elements of matrix B:
1 1 1 1
Enter number of rows and columns for matrix C: 2 2
Enter elements of matrix C:
1 1 1 1
Enter number of rows and columns for matrix D: 2 2
Enter elements of matrix D:
1 1 1 1
Resulting matrix T:
18 18
18 18
```

=== Code Execution Successful ===

Conclusion:

This C program performs various matrix operations including multiplication, transposition, and exponentiation. It reads four matrices (A, B, C, D) from the user and calculates intermediate results such as the product of A and B, the product of C and D, and the square of CD. It then computes the transpose of CD squared and the fourth power of matrix A. Finally, the program combines these results to produce and display the resulting matrix T, demonstrating comprehensive matrix manipulation.