Winter Winning Camp(Assignment-1)

Student Name: Shreya Biswas **UID: 22BCS15325**
Semester: 5TH **Section: 620-A**
Subject: DSA(C++) **Branch: BE-CSE**

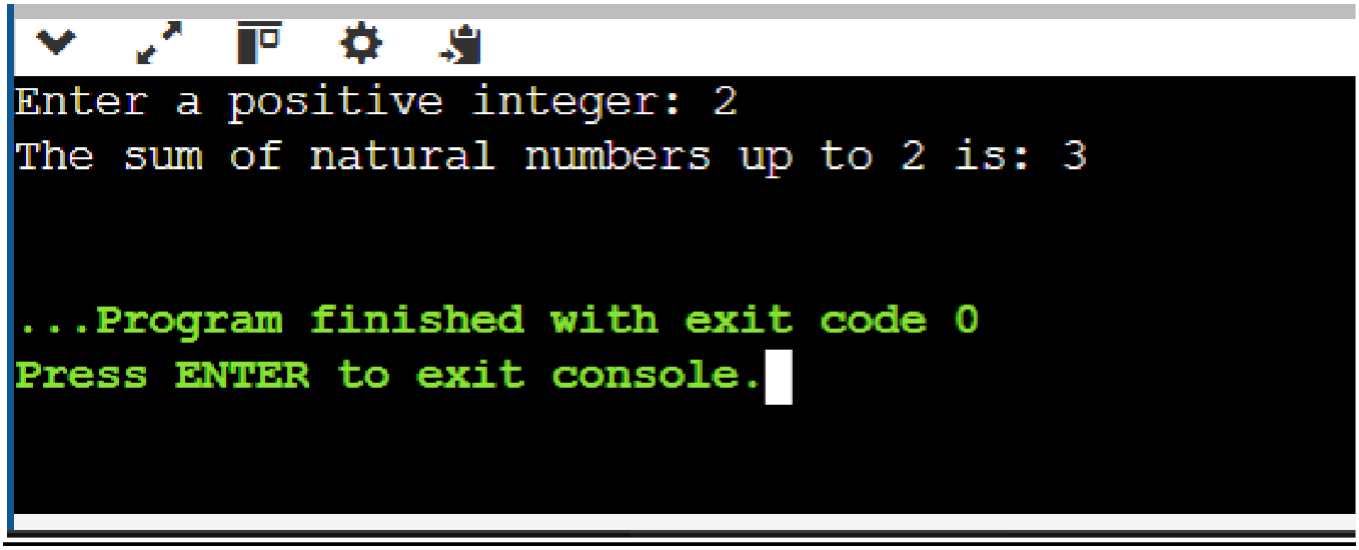
1) Sum of Natural Numbers up to N

Calculate the sum of all natural numbers from 1 to n, where n is a positive integer. Use the formula:
Sum=n×(n+1)/2 .
Take n as input and output the sum of natural numbers from 1 to n .

Code:-

```
#include  <iostream>
using namespace std;
int main() {
    int n;

    // Input the value of n
    cout << "Enter a positive integer: ";
    cin >> n;

    // Check if n is positive
    if (n > 0) {
        // Calculate the sum using the formula
        int sum = n * (n + 1) / 2;

        // Output the result
        cout << "The sum of natural numbers up to " << n << " is: " << sum << endl;
    } else {
        cout << "Please enter a positive integer." << endl;
    }

    return 0;
     }
```

OUTPUT:-



2) Check if a Number is Prime

Objective

Check if a given number n is a prime number. A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.

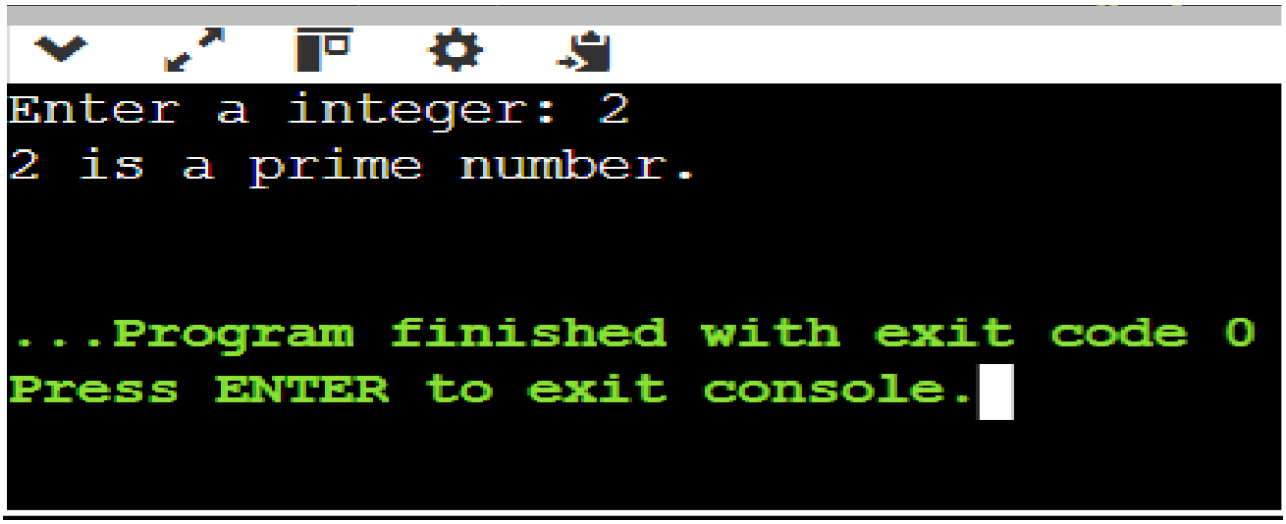
To determine if a number is prime, iterate from 2 to √n and check if n is divisible by any number in this range. If it is divisible, it is not a prime number; otherwise, it is a prime.

Code:-

```
#include <iostream>
#include <cmath>
using namespace std;

int main() {
    int n;


    cout << "Enter a integer: ";
    cin >> n;
```

```
    if (n <= 1) {
        cout << n << " is not a prime number." << endl;
    } else {
        bool isPrime = true;

        for (int i = 2; i <= sqrt(n); i++) {
            if (n % i == 0) {
                isPrime = false;
                break;
            }
        }

        if (isPrime) {
            cout << n << " is a prime number." << endl;
        } else {
            cout << n << " is not a prime number." << endl;
        }
    }

    return 0;
}
```

OUTPUT:-



3) Print Multiplication Table of a Number

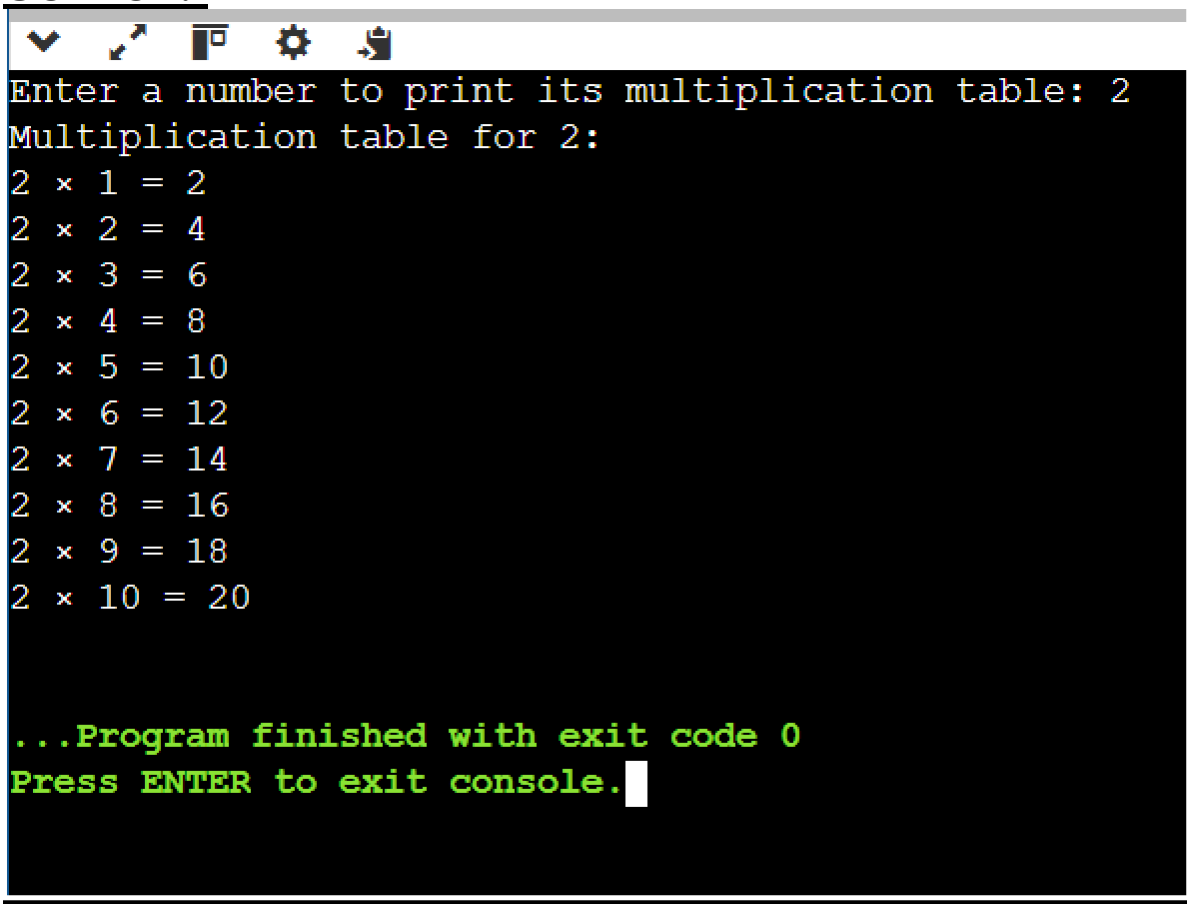
Objective

Print the multiplication table of a given number n. A multiplication table for a number n is a list of products of n with integers from 1 to 10. For example, the multiplication table for 3 is:
3×1=3,3×2=6,…,3×10=30.

Code:-

```
#include <iostream>
using namespace std;

int main() {
    int n;

    cout << "Enter a number to print its multiplication table: ";
    cin >> n;
    cout << "Multiplication table for " << n << ":" << endl;
    for (int i = 1; i <= 10; i++) {
        cout << n << " × " << i << " = " << n * i << endl;
    }

    return 0;
}
```

OUTPUT:-



4) Check if a Number is a Palindrome

Objective

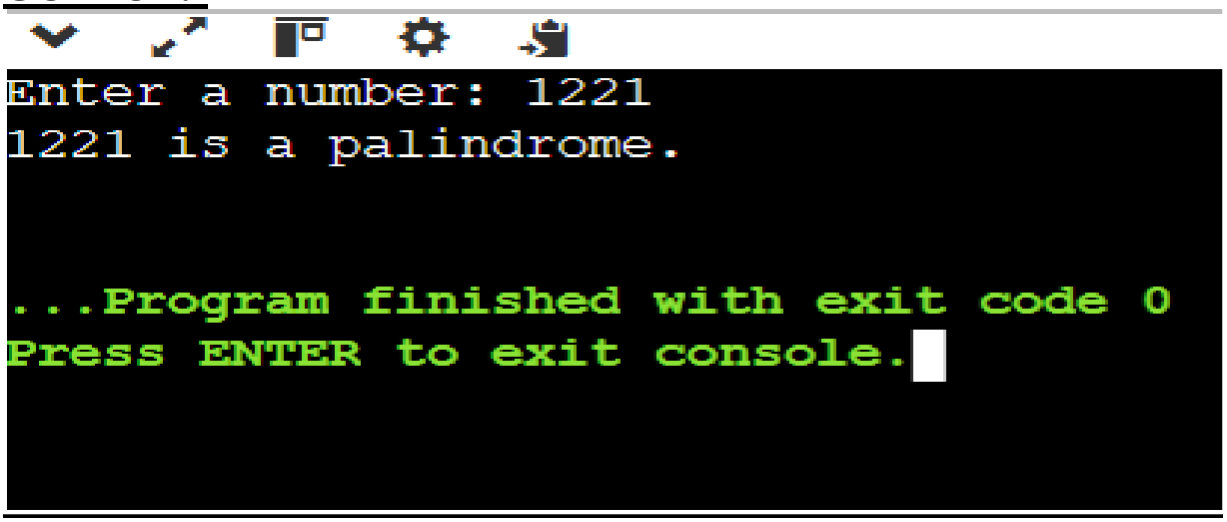
Check whether a given number is a palindrome or not. A number is called a palindrome if it reads the same backward as forward. For example, 121 is a palindrome because reading it from left to right is the same as reading it from right to left. Similarly, 12321 is also a palindrome, but 12345 is not.

Code:-

```
#include <iostream>

using namespace std;
```

```
int main() {
    int n, original, reversed = 0, remainder;

    cout << "Enter a number: ";
    cin >> n;

    original = n; // Store the original number

    while (n != 0) {
        remainder = n % 10;
        reversed = reversed * 10 + remainder;
        n /= 10;
    }

    if (original == reversed) {
        cout << original << " is a palindrome." << endl;
    } else {
        cout << original << " is not a palindrome." << endl;
    }
    return 0;
}
```

OUTPUT:-



5) Function Overloading for Calculating Area.

Objective

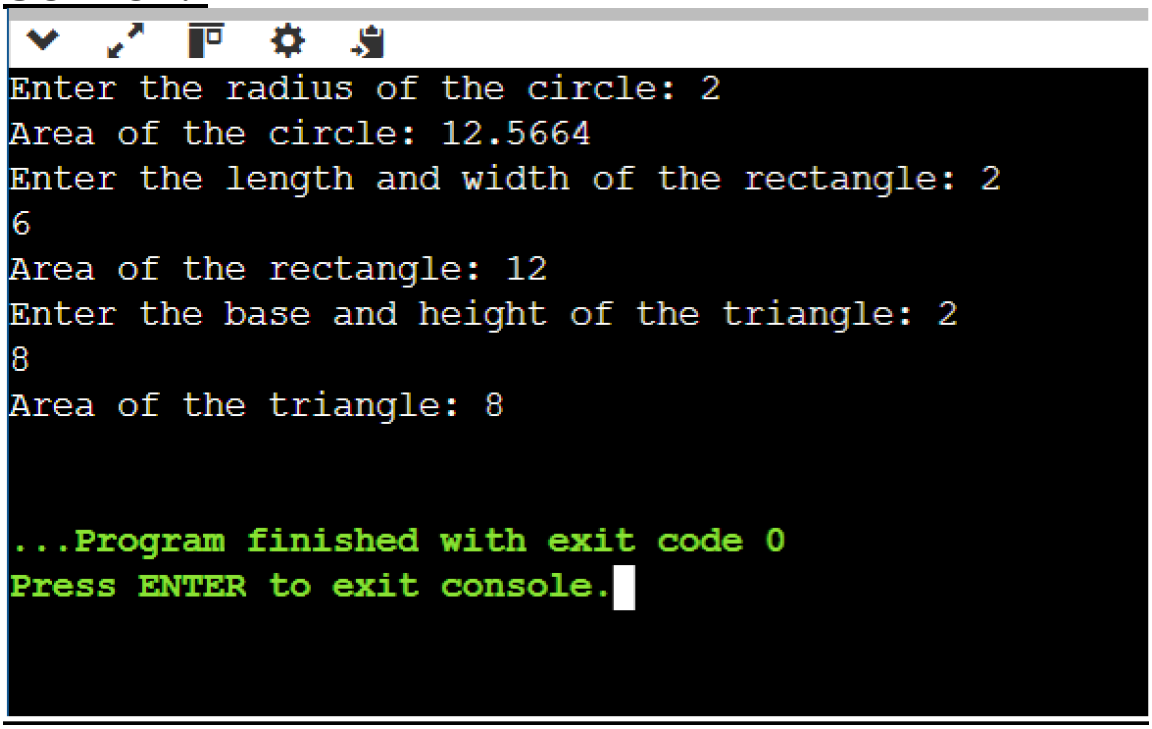
Write a program to calculate the area of different shapes using function overloading. Implement overloaded functions to compute the area of a circle, a rectangle, and a triangle.

Code:-

```
#include <iostream>
#include <cmath>
using namespace std;

double area(double radius) {
    return M_PI * radius * radius; // π * r^2
}

double area(double length, double width) {
    return length * width; // length * width
}

double area(double base, double height, int) {
    return 0.5 * base * height; // 0.5 * base * height
}
```

```
int main() {
    double radius, length, width, base, height;

    // Circle
    cout << "Enter the radius of the circle: ";
    cin >> radius;
    cout << "Area of the circle: " << area(radius) << endl;

    // Rectangle
    cout << "Enter the length and width of the rectangle: ";
    cin >> length >> width;
    cout << "Area of the rectangle: " << area(length, width) << endl;

    // Triangle
    cout << "Enter the base and height of the triangle: ";
    cin >> base >> height;
    cout << "Area of the triangle: " << area(base, height, 0) << endl;

    return 0;
}
```

OUTPUT:-



6) Function Overloading with Hierarchical Structure.

Objective

Write a program that demonstrates function overloading to calculate the salary of employees at different levels in a company hierarchy.
Implement overloaded functions to compute salary for:

- **Intern (basic stipend).**
- **Regular employee (base salary + bonuses).**
- **Manager (base salary + bonuses + performance incentives).**

Code:-

```
#include <iostream>
using namespace std;

double calculateSalary(double stipend) {
    return stipend;
}

double calculateSalary(double baseSalary, double bonus) {
    return baseSalary + bonus;
}

double calculateSalary(double baseSalary, double bonus, double incentives) {
    return baseSalary + bonus + incentives;
}

int main() {
    double stipend, baseSalary, bonus, incentives;

    cout << "Enter the stipend for the intern: ";
    cin >> stipend;
    cout << "Intern's salary: " << calculateSalary(stipend) << endl;

    cout << "Enter the base salary and bonus for the regular employee: ";
    cin >> baseSalary >> bonus;
    cout << "Regular employee's salary: " << calculateSalary(baseSalary,
```

```
bonus) << endl;

    cout << "Enter the base salary, bonus, and incentives for the manager: ";
    cin >> baseSalary >> bonus >> incentives;
    cout << "Manager's salary: " << calculateSalary(baseSalary, bonus,
incentives) << endl;

    return 0;
}
```

OUTPUT:-

```
input
Enter the stipend for the intern: 5000
Intern's salary: 5000
Enter the base salary and bonus for the regular employee: 3000 5000
Regular employee's salary: 8000
Enter the base salary, bonus, and incentives for the manager: 50000 10000 15000
Manager's salary: 75000


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

7) Inheritance with Student and Result Classes.

Objective

Create a program that demonstrates inheritance by defining:

- **A base class Student to store details like Roll Number and Name.**
- **A derived class Result to store marks for three subjects and calculate the total and percentage.**

Code:-

```
#include <iostream>
#include <string>
using namespace std;

class Student {
protected:
    int rollNumber;
```

```
    string name;

public:
    void inputDetails() {
        cout << "Enter Roll Number: ";
        cin >> rollNumber;
        cin.ignore(); // Clear input buffer
        cout << "Enter Name: ";
        getline(cin, name);
    }

    void displayDetails() {
        cout << "Roll Number: " << rollNumber << endl;
        cout << "Name: " << name << endl;
    }
};

class Result : public Student {
private:
    float marks[3];
    float total;
    float percentage;

public:
    void inputMarks() {
        cout << "Enter marks for three subjects:" << endl;
        total = 0;
        for (int i = 0; i < 3; i++) {
            cout << "Subject " << i + 1 << ": ";
            cin >> marks[i];
            total += marks[i];
        }
        percentage = total / 3.0;
    }

    void displayResult() {
        cout << "Marks in Subjects: ";
```

```
        for (int i = 0; i < 3; i++) {
            cout << marks[i] << " ";
        }
        cout << endl;
        cout << "Total Marks: " << total << endl;
        cout << "Percentage: " << percentage << "%" << endl;
    }
};

int main() {
    Result student;

    student.inputDetails();
    student.inputMarks();

    cout << "\nStudent Details and Results:" << endl;
    student.displayDetails();
    student.displayResult();

    return 0;
}
```

OUTPUT:-

```
Enter Roll Number: 36
Enter Name: Robert Downey Jr.
Enter marks for three subjects:
Subject 1: 38
Subject 2: 39
Subject 3: 40

Student Details and Results:
Roll Number: 36
Name: Robert Downey Jr.
Marks in Subjects: 38 39 40
Total Marks: 117
Percentage: 39%


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

8) Reverse a Number

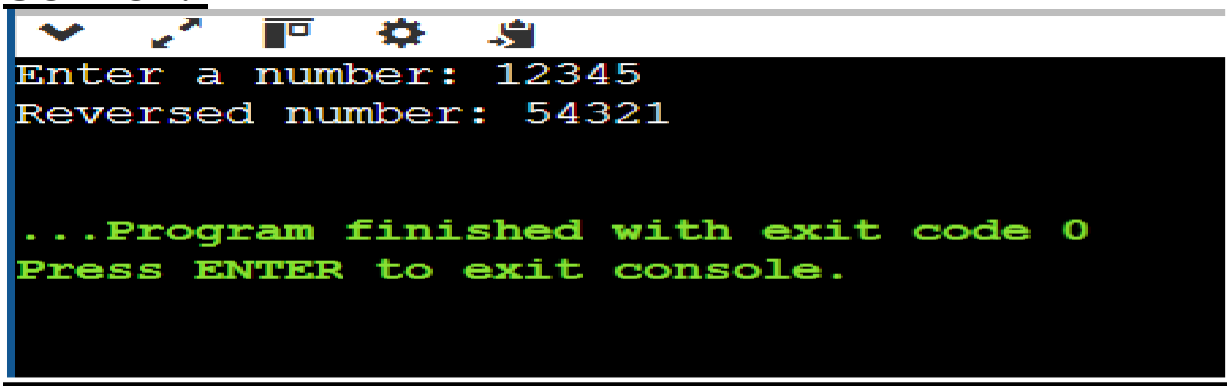
Objective

Reverse the digits of a given number n. For example, if the input number is 12345, the output should be 54321. The task involves using loops and modulus operators to extract the digits and construct the reversed number.

Code:-

```
#include <iostream>
using namespace std;

int main() {
    int n, reversed = 0, remainder;


    cout << "Enter a number: ";
    cin >> n;

    while (n != 0) {
        remainder = n % 10;
        reversed = reversed * 10 + remainder;
        n /= 10;
    }
    cout << "Reversed number: " << reversed << endl;

    return 0;
}
```

OUTPUT:-



9) Matrix Multiplication Using Function Overloading

Objective

Implement matrix operations in C++ using function overloading. Write a function operate() that can perform:

- Matrix Addition **for matrices of the same dimensions.**
- Matrix Multiplication **where the number of columns of the first matrix equals the number of rows of the second matrix.**

Code:-

```
#include <iostream>
#include <vector>
using namespace std;

vector<vector<int>> operate(const vector<vector<int>>& a, const
vector<vector<int>>& b) {
    vector<vector<int>> result(a.size(), vector<int>(a[0].size()));
    for (int i = 0; i < a.size(); i++) {
        for (int j = 0; j < a[0].size(); j++) {
            result[i][j] = a[i][j] + b[i][j];
        }
    }
    return result;
}

vector<vector<int>> operate(const vector<vector<int>>& a, const
vector<vector<int>>& b, int) {
    vector<vector<int>> result(a.size(), vector<int>(b[0].size(), 0));
    for (int i = 0; i < a.size(); i++) {
        for (int j = 0; j < b[0].size(); j++) {
            for (int k = 0; k < a[0].size(); k++) {
                result[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return result;
}
```

```
void displayMatrix(const vector<vector<int>>& matrix) {
    for (const auto& row : matrix) {
        for (int elem : row) {
            cout << elem << " ";
        }
        cout << endl;
    }
}

int main() {
    int rows1, cols1, rows2, cols2;

    cout << "Enter rows and columns for first matrix: ";
    cin >> rows1 >> cols1;
    cout << "Enter rows and columns for second matrix: ";
    cin >> rows2 >> cols2;

    if ((rows1 != rows2 || cols1 != cols2) && cols1 != rows2) {
        cout << "Matrix operation not possible due to incompatible dimensions!"
<< endl;
        return 0;
    }

    vector<vector<int>> mat1(rows1, vector<int>(cols1));
    vector<vector<int>> mat2(rows2, vector<int>(cols2));

    cout << "Enter elements for first matrix:" << endl;
    for (int i = 0; i < rows1; i++) {
        for (int j = 0; j < cols1; j++) {
            cin >> mat1[i][j];
        }
    }

    cout << "Enter elements for second matrix:" << endl;
    for (int i = 0; i < rows2; i++) {
        for (int j = 0; j < cols2; j++) {
```

```
            cin >> mat2[i][j];
        }
    }


    if (rows1 == rows2 && cols1 == cols2) {
        cout << "Matrix Addition Result:" << endl;
        displayMatrix(operate(mat1, mat2));  // Add matrices
    } else {
        cout << "Matrix addition not possible (dimensions do not match)." <<
endl;
    }


    if (cols1 == rows2) {
        cout << "Matrix Multiplication Result:" << endl;
        displayMatrix(operate(mat1, mat2, 0));  // Multiply matrices
    } else {
        cout << "Matrix multiplication not possible (columns of first matrix must
equal rows of second matrix)." << endl;
    }

    return 0;
}
```

OUTPUT:-

```
Enter rows and columns for first matrix: 2
2
Enter rows and columns for second matrix: 2
2
Enter elements for first matrix:
1
2
3
3
Enter elements for second matrix:
3
5
6
4
Matrix Addition Result:
4 7
9 7
Matrix Multiplication Result:
15 13
27 27


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

10) Implementing Polymorphism for Shape Hierarchies.

Objective

Write a program to demonstrate runtime polymorphism in C++ using a base class Shape and derived classes Circle, Rectangle, and Triangle. The program should use virtual functions to calculate and print the area of each shape based on user input.

Code:-

```
#include <iostream>
#include <cmath>
using namespace std;

class Shape {
public:
    virtual void area() = 0; // Pure virtual function
    virtual ~Shape() {} // Virtual destructor
};

class Circle : public Shape {
    double radius;
public:
    Circle(double r) : radius(r) {}
    void area() override {
        cout << "Area of Circle: " << M_PI * radius * radius << endl;
    }
};

class Rectangle : public Shape {
    double length, width;
public:
    Rectangle(double l, double w) : length(l), width(w) {}
    void area() override {
        cout << "Area of Rectangle: " << length * width << endl;
    }
};
```

```
class Triangle : public Shape {
    double base, height;
public:
    Triangle(double b, double h) : base(b), height(h) {}
    void area() override {
        cout << "Area of Triangle: " << 0.5 * base * height << endl;
    }
};

int main() {
    Shape* shapes[] = { new Circle(5), new Rectangle(4, 6), new Triangle(4, 7)
};

    for (auto shape : shapes) {
        shape->area();
        delete shape; // Clean up
    }

    return 0;
}
```

OUTPUT:-