Rahul

22BCS15287

IOT-620(A)   <u>**Winter winning camp**</u>

<u>Day-01</u>

Very Easy

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 .

Task

Given an integer n, print the sum of all natural numbers from 1 to n.

CODE:

```cpp
#include <iostream>

using namespace std;

int main() {

    int n;

    cout << "Enter a positive integer: ";

    cin >> n;

    if (n <= 0) {

        cout << "Please enter a positive integer." << endl;

        return 1;
```

```
    }

    int sum = n * (n + 1) / 2;

    cout << "The sum of natural numbers from 1 to " << n << " is: " << sum << endl;

    return 0;

}
```

Output:


```
Output
Enter a positive integer: 1000
The sum of natural numbers from 1 to 1000 is: 500500
```

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.

Task

Given an integer n, print "Prime" if the number is prime, or "Not Prime" if it is not.

CODE:
```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main() {
    int n;
    cout << "Enter a positive integer: ";
    cin >> n;

    if (n <= 1) {
        cout << n << " is not a prime number." << endl;
```

```cpp
        return 0;
    }
    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;
}
```
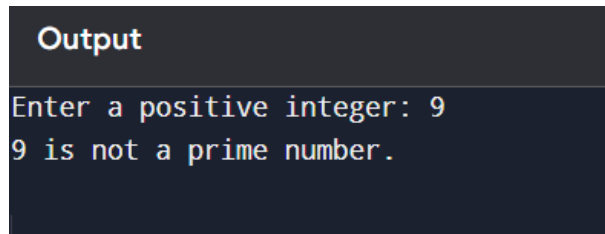
OUTPUT:

```
Output

Enter a positive integer: 9
9 is not a prime number.
```

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: $3\times1=3, 3\times2=6,\ldots,3\times10=30$.

Task

Given an integer n, print the multiplication table of n from $1\times n$ to $10\times n$.

Code:

```cpp
#include <iostream>

using namespace std;

int main() {
```

```cpp
    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 << " x " << i << " = " << n * i << endl;

    }

    return 0;

}
```

Output:

```
Output
Enter a number to print its multiplication table: 7
Multiplication table for 7:
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

Easy:

4) Count Digits in a Number

Objective

Count the total number of digits in a given number n. The number can be a positive integer. For example, for the number 12345, the count of digits is 5. For a number like 900000, the count of digits is 6.

Given an integer n, your task is to determine how many digits are present in n. This task will help you practice working with loops, number manipulation, and conditional logic.

Task

Given an integer n, print the total number of digits in n.

Code:

```cpp
#include <iostream>

using namespace std;

int main() {

    int n;

    cout << "Enter a positive integer: ";

    cin >> n;

    if (n == 0) {

        cout << "The number of digits in 0 is 1." << endl;

        return 0;

    }

    int count = 0;

    while (n != 0) {

        n /= 10; // Remove the last digit

        count++;

    }

    cout << "The total number of digits is: " << count << endl;

    return 0;

}
```

Output:

```
Output

Enter a positive integer: 12345
The total number of digits is: 5
```

5) 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.

Task

Given an integer n, print the number with its digits in reverse order.

Code:

```cpp
#include <iostream>

using namespace std;

int main() {

    int n, reversedNumber = 0;

    cout << "Enter a positive integer: ";

    cin >> n;


    while (n != 0) {

        int digit = n % 10;

        reversedNumber = reversedNumber * 10 + digit;

        n /= 10;

    }
```

cout << "The reversed number is: " << reversedNumber << endl;

   return 0;

}

```
Output
Enter a positive integer: 96751
The reversed number is: 15769
```

6) Find the Largest Digit in a Number

Objective

Find the largest digit in a given number n. For example, for the number 2734, the largest digit is 7. You need to extract each digit from the number and determine the largest one. The task will involve using loops and modulus operations to isolate the digits.

Task

Given an integer n, find and print the largest digit in n.

Code:

```
#include <iostream>

using namespace std;

int main() {

   int n;

   cout << "Enter a positive integer: ";

   cin >> n;

   if (n <= 0) {

      cout << "Please enter a positive integer." << endl;

      return 1;

   }
```

```cpp
    int largestDigit = 0;

    while (n != 0) {

        int digit = n % 10; // Extract the last digit

        if (digit > largestDigit) {

            largestDigit = digit;

        }

        n /= 10; // Remove the last digit

    }

    cout << "The largest digit is: " << largestDigit << endl;

    return 0;

}
```

```
Output

Enter a positive integer: 45367
The largest digit is: 7
```

7) Find the Sum of Digits of a Number

Objective

Calculate the sum of the digits of a given number n. For example, for the number 12345, the sum of the digits is 1+2+3+4+5=15. To solve this, you will need to extract each digit from the number and calculate the total sum.

Task

Given an integer n, find and print the sum of its digits.

Code:

```cpp
#include <iostream>
```

```cpp
using namespace std;

int main() {
    int n;
    cout << "Enter a positive integer: ";
    cin >> n;
    if (n < 0) {
        cout << "Please enter a positive integer." << endl;
        return 1;
    }

    int sum = 0;
    while (n != 0) {
        int digit = n % 10; // Extract the last digit
        sum += digit;       // Add the digit to the sum
        n /= 10;            // Remove the last digit
    }
    cout << "The sum of the digits is: " << sum << endl;
    return 0;
}
```

```
Output
Enter a positive integer: 345745
The sum of the digits is: 28
```

Medium:

8) 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.

```cpp
Code:
#include <iostream>
#include <cmath>
using namespace std;

double calculateArea(double radius) {
    return M_PI * radius * radius;
}
double calculateArea(double length, double width) {
    return length * width;
}
double calculateArea(double base, double height, bool isTriangle) {
    return 0.5 * base * height;
}

int main() {
    int choice;
```

```cpp
    cout << "Choose the shape to calculate the area:" << endl;
    cout << "1. Circle" << endl;
    cout << "2. Rectangle" << endl;
    cout << "3. Triangle" << endl;
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
        case 1: {
            double radius;
            cout << "Enter the radius of the circle: ";
            cin >> radius;
            cout << "The area of the circle is: " << calculateArea(radius) << endl;
            break;
        }
        case 2: {
            double length, width;
            cout << "Enter the length and width of the rectangle: ";
            cin >> length >> width;
            cout << "The area of the rectangle is: " << calculateArea(length, width) << endl;
            break;
        }
        case 3: {
            double base, height;
            cout << "Enter the base and height of the triangle: ";
            cin >> base >> height;
            cout << "The area of the triangle is: " << calculateArea(base, height, true) << endl;
            break;
        }
        default:
            cout << "Invalid choice!" << endl;
    }

    return 0;
}
```

```
 Output

Choose the shape to calculate the area:
1. Circle
2. Rectangle
3. Triangle
Enter your choice: 3
Enter the base and height of the triangle: 5
8
The area of the triangle is: 20
```

9) Polymorphism with Shape Area Calculation.

Objective

Create a program that demonstrates polymorphism by calculating the area of different shapes using a base class Shape and derived classes for Circle, Rectangle, and Triangle. Each derived class should override a virtual function to compute the area of the respective shape.

Input Format

The program should accept:

1. Radius for a circle.
2. Length and breadth for a rectangle.
3. Base and height for a triangle.

Code:
```cpp
#include <iostream>
#include <cmath>
using namespace std;
class Shape {
public:
    virtual double calculateArea() = 0; // Pure virtual function
    virtual ~Shape() {}
};
class Circle : public Shape {
private:
    double radius;
public:
    Circle(double r) : radius(r) {}
    double calculateArea() override {
        return M_PI * radius * radius;
    }
};
class Rectangle : public Shape {
private:
    double length, breadth;
public:
    Rectangle(double l, double b) : length(l), breadth(b) {}
    double calculateArea() override {
        return length * breadth;
    }
};
class Triangle : public Shape {
private:
    double base, height;
public:
    Triangle(double b, double h) : base(b), height(h) {}
```

```cpp
        double calculateArea() override {
            return 0.5 * base * height;
        }
    };

    int main() {
        Shape* shape = nullptr;
        int choice;

        cout << "Choose the shape to calculate the area:" << endl;
        cout << "1. Circle" << endl;
        cout << "2. Rectangle" << endl;
        cout << "3. Triangle" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                double radius;
                cout << "Enter the radius of the circle: ";
                cin >> radius;
                shape = new Circle(radius);
                break;
            }
            case 2: {
                double length, breadth;
                cout << "Enter the length and breadth of the rectangle: ";
                cin >> length >> breadth;
                shape = new Rectangle(length, breadth);
                break;
            }
            case 3: {
                double base, height;
                cout << "Enter the base and height of the triangle: ";
                cin >> base >> height;
                shape = new Triangle(base, height);
                break;
            }
            default:
                cout << "Invalid choice!" << endl;
                return 1;
        }

        if (shape) {
            cout << "The area of the chosen shape is: " << shape->calculateArea() << endl;
            delete shape;
```

```
    }
    return 0;
}
```

Hard:

10) 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.

Input Format

- **Matrix A**: A 2D matrix of dimensions m× n.
- **Matrix B**: A 2D matrix of dimensions n× p.
- An integer indicating the operation type:
    - 1 for Matrix Addition.
    - 2 for Matrix Multiplication.

Code:
```
#include <iostream>
#include <vector>

using namespace std;

// Function to perform matrix addition
```

```cpp
vector<vector<int>> operate(const vector<vector<int>>& A, const vector<vector<int>>& B, int
operation) {
    int m = A.size();
    int n = A[0].size();
    int p = B[0].size();

    if (operation == 1) { // Matrix Addition
        if (m != B.size() || n != p) {
            throw invalid_argument("Invalid dimensions for operation");
        }
        vector<vector<int>> result(m, vector<int>(n, 0));
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                result[i][j] = A[i][j] + B[i][j];
            }
        }
        return result;
    } else if (operation == 2) { // Matrix Multiplication
        if (n != B.size()) {
            throw invalid_argument("Invalid dimensions for operation");
        }
        vector<vector<int>> result(m, vector<int>(p, 0));
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < p; ++j) {
                for (int k = 0; k < n; ++k) {
                    result[i][j] += A[i][k] * B[k][j];
                }
            }
        }
        return result;
    } else {
        throw invalid_argument("Unknown operation");
    }
}

int main() {
    vector<vector<int>> A = {{1, 2}, {3, 4}};
    vector<vector<int>> B = {{5, 6}, {7, 8}};
    int operation = 1; // Change to 2 for matrix multiplication

    try {
        vector<vector<int>> result = operate(A, B, operation);

        cout << "Result:" << endl;
        for (const auto& row : result) {
            for (const auto& elem : row) {
                cout << elem << " ";
            }
        }
```

```cpp
            cout << endl;
        }
    } catch (const exception& e) {
        cout << e.what() << endl;
    }

    return 0;
}
```

```
Output

Result:
6  8
10  12
```