Domain winter winning Camp
DAY-1

Name: -Priti

Section/Group: -Kpit-901-B

Topic: C++ Basic & Input/Output:

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.

Input Format

One integer n, the upper limit for calculating the sum.

Constraints

- $1 \leq n \leq 104$.

Output Format

Print the sum of all natural numbers from 1 to n.

Test Cases:

Example 1 Input:

5

Output:

15

Explanation:

Using the formula, Sum=5×(5+1)/2=15 .

Example 2 Input:

100

Output:

5050

Explanation:

Using the formula, Sum=100×(100+1)/2=5050 .

Example 3 Input:

1

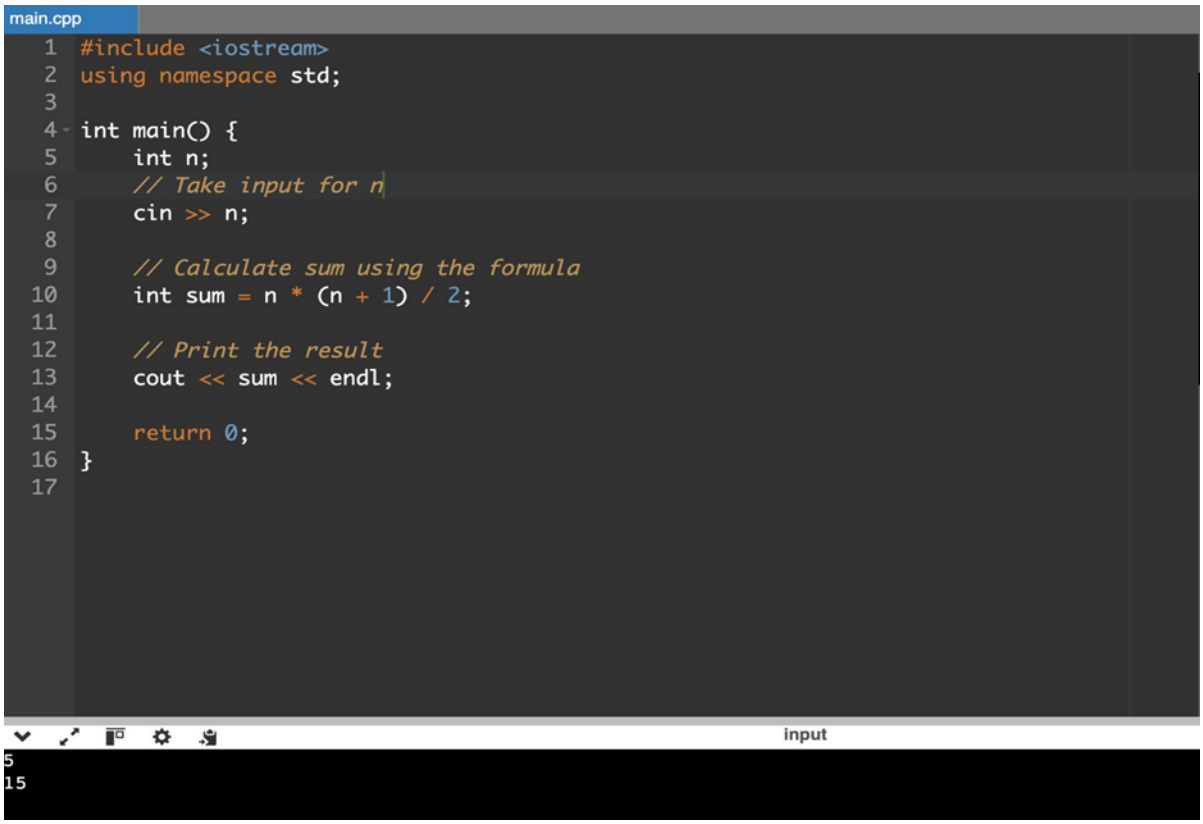
Output:

1

Explanation:

Using the formula, Sum=1×(1+1)/2=1 .

SOLUTION:



Easy:

1) Count Digits in a Number Objective

Count the total number of digits in each 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.

Input Format One integer n. Constraints

- $1 \leq n \leq 109$

Output Format

Print the number of digits in n.

Test Cases Example 1:

Input:

12345

Output:

5

Explanation:

The number 12345 has 5 digits: 1, 2, 3, 4, 5.

Example 2: Input:

900000

Output:

6

Explanation:

The number 900000 has 6 digits: 9, 0, 0, 0, 0, 0.

Example 3:

Input:

1

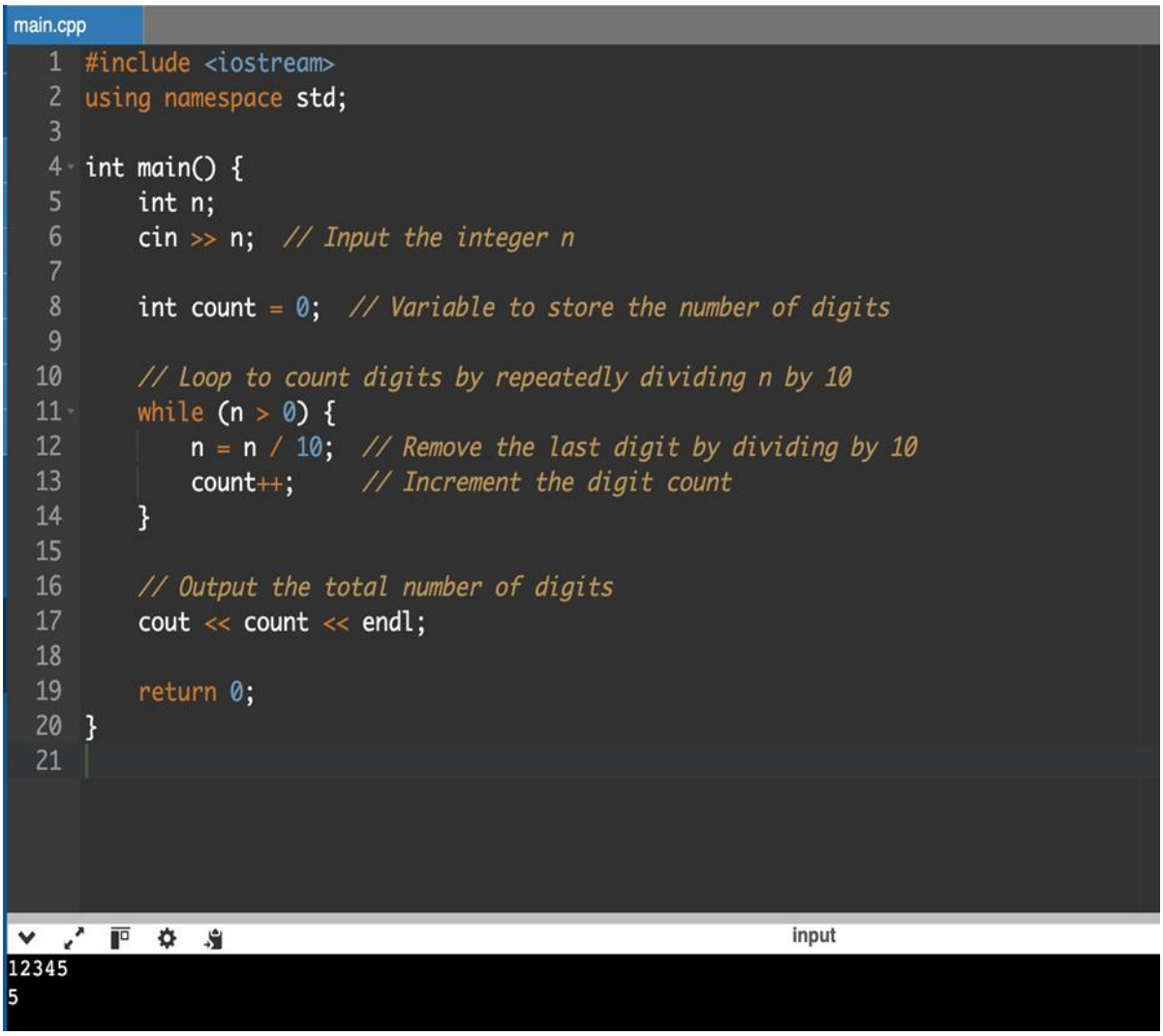
Output:

1

Explanation:

The number 1 has only 1 digit.

SOLUTION:



Medium:

1) 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.

Input Format

The program should accept:

1. Radius of the circle for the first function.
2. Length and breadth of the rectangle for the second function.
3. Base and height of the triangle for the third function.

Constraints

$1 \leq$ radius, length, breadth, base, height ≤ 103

Use 3.14159 for the value of π.

Output Format

Print the computed area of each shape in a new line.

Test Cases:

Example 1 Input:

Radius = 5

Length = 4, breadth = 6

Base = 3, height = 7

Output:

78.53975

24

10.5

Explanation:

- The area of the circle with radius 5 is 3.14159* 52 = 78.53975.
- The area of the rectangle with length 4 and breadth 6 is 4* 6 = 24.
- The area of the triangle with base 3 and height 7 is 0.5*3*7 = 10.5.

Example 2 Input:

Radius = 10

Length = 15, breadth = 8

Base = 12, height = 9

Output:

314.159

120

54

Explanation:

- The area of the circle with radius 10 is 3.14159 * 102 = 314.159.
- The area of the rectangle with length 15 and breadth 8 is 15* 8 = 120.
- The area of the triangle with base 12 and height 9 is 0.5* 12 * 9 = 54.

Example 3 Input:

Radius = 1

length = 2, breadth = 3

Base = 5, height = 8

Output:

3.14159

6

20

Explanation:

The area of the circle with radius 1 is 3.14159 * 12 = 3.14159 . The area of the rectangle with length 2 and breadth 3 is 2* 3 = 6.

The area of the triangle with base 5 and height 8 is 0.5 *5 * 8 = 20.

SOLUTION:

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

// Function to calculate the area of a circle
double calculateArea(double radius) {
    return 3.14159 * radius * radius;
}

// Function to calculate the area of a rectangle
int calculateArea(int length, int breadth) {
    return length * breadth;
}

// Function to calculate the area of a triangle
double calculateArea(double base, double height) {
    return 0.5 * base * height;
}

int main() {
    // Input values
    double radius, base, height;
    int length, breadth;

    // Reading input
    cin >> radius;
    cin >> length >> breadth;
    cin >> base >> height;

    // Calling overloaded functions and printing results
    cout << calculateArea(radius) << endl;  // Area of circle
    cout << calculateArea(length, breadth) << endl;  // Area of rectangle
    cout << calculateArea(base, height) << endl;  // Area of triangle

    return 0;
}
```

Input

```
3
24
30
2
4
28.2743
720
4
```

Hard

1) Implement Polymorphism for Banking Transactions Objective

Design a C++ program to simulate a banking system using polymorphism. Create a base class Account with a virtual method calculateInterest(). Use the derived classes SavingsAccount and CurrentAccount to implement specific interest calculation logic:

- SavingsAccount: Interest = Balance × Rate × Time.
- CurrentAccount: No interest, but includes a maintenance fee deduction.

Input Format

1. Account Type (1 for Savings, 2 for Current).
2. Account Balance (integer).
3. For Savings Account: Interest Rate (as a percentage) and Time (in years).
4. For Current Account: Monthly Maintenance Fee.

Constraints

- Account type: $1 \leq \text{type} \leq 2$.
- Balance: $1000 \leq \text{balance} \leq 1{,}000{,}000$.
- Interest Rate: $1 \leq \text{rate} \leq 15$.
- Time: $1 \leq \text{time} \leq 10$.
- Maintenance Fee: $50 \leq \text{fee} \leq 500$.

Test Cases:

Example 1: Savings Account Interest Input:

Account Type: 1

Balance: 10000

Interest Rate: 5

Time: 3

Output:

Savings Account Interest: 1500

Example 2: Current Account Fee Input:

Account Type: 2

Balance: 20000

Maintenance Fee: 200

Output:

Balance after fee deduction: 19800

Example 3: Invalid Account Type Input:

Account Type: 3

Output:

Invalid account type.

SOLUTION:

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

// Base class Account
class Account {
protected:
    double balance;

public:
    Account(double balance) : balance(balance) {}

    // Virtual method for calculating interest or fee (to be overridden in derived classes)
    virtual void calculateInterestOrFee() = 0;

    // Method to get the current balance
    double getBalance() {
        return balance;
    }
};

// Derived class for SavingsAccount
class SavingsAccount : public Account {
private:
    double interestRate;
    int time;

public:
    SavingsAccount(double balance, double interestRate, int time)
        : Account(balance), interestRate(interestRate), time(time) {}

    // Overriding the calculateInterestOrFee method to calculate interest for SavingsAccount
    void calculateInterestOrFee() override {
        double interest = balance * (interestRate / 100) * time;
        cout << "Savings Account Interest: " << interest << endl;
    }
};

// Derived class for CurrentAccount
class CurrentAccount : public Account {
private:
    double maintenanceFee;

public:
    CurrentAccount(double balance, double maintenanceFee)
        : Account(balance), maintenanceFee(maintenanceFee) {}

    // Overriding the calculateInterestOrFee method to deduct maintenance fee for CurrentAccount
    void calculateInterestOrFee() override {
        CurrentAccount(double balance, double maintenanceFee)
            : Account(balance), maintenanceFee(maintenanceFee) {}

        // Overriding the calculateInterestOrFee method to deduct maintenance fee for CurrentAccount
        void calculateInterestOrFee() override {
            balance -= maintenanceFee;
            cout << "Balance after fee deduction: " << balance << endl;
        }
};

int main() {
    int accountType;
    double balance, interestRate, maintenanceFee;
    int time;

    // Input: Account type, balance and other parameters based on account type
    cout << "Enter account type (1 for Savings, 2 for Current): ";
    cin >> accountType;

    if (accountType == 1) {  // Savings Account
        cout << "Enter balance: ";
        cin >> balance;
        cout << "Enter interest rate (in %): ";
        cin >> interestRate;
        cout << "Enter time (in years): ";
        cin >> time;

        // Create SavingsAccount object and calculate interest
        SavingsAccount savingsAccount(balance, interestRate, time);
        savingsAccount.calculateInterestOrFee();

    } else if (accountType == 2) {  // Current Account
        cout << "Enter balance: ";
        cin >> balance;
        cout << "Enter monthly maintenance fee: ";
        cin >> maintenanceFee;

        // Create CurrentAccount object and calculate fee deduction
        CurrentAccount currentAccount(balance, maintenanceFee);
        currentAccount.calculateInterestOrFee();

    } else {
        cout << "Invalid account type." << endl;  // Invalid account type
    }

    return 0;
```

```
input
Enter account type (1 for Savings, 2 for Current): 1
Enter balance: 34567
Enter interest rate (in %): 6
Enter time (in years): 3
Savings Account Interest: 6222.06


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

Very Hard

1) Hierarchical Inheritance for Employee Management System Objective

Create a C++ program to simulate an employee management system using hierarchical inheritance. Design a base class Employee that stores basic details (name, ID, and salary). Create two derived classes:

Manager: Add and calculate bonuses based on performance ratings.

Developer: Add and calculate overtime compensation based on extra hours worked.

The program should allow input for both types of employees and display their total earnings.

Input Format

1. Employee Type (1 for Manager, 2 for Developer).
2. Name (string), ID (integer), and salary (integer).
3. For Manager: Performance Rating (1–5).
4. For Developer: Extra hours worked (integer).

Constraints

- Employee type: $1 \leq \text{type} \leq 2$.
- Salary: $10{,}000 \leq \text{salary} \leq 1{,}000{,}000$.
- Rating: $1 \leq \text{rating} \leq 5$.
- Extra hours: $0 \leq \text{hours} \leq 100$.
- Bonus per rating point: 10% of salary.
- Overtime rate: $500 per hour.

Test Cases:

Example 1: Manager with Rating Bonus Input:

Employee Type: 1 Name: Alice

ID: 101

Salary: 50000

Rating: 4

Output:

Employee: Alice (ID: 101) Role: Manager

Base Salary: 50000

Bonus: 20000

Total Earnings: 70000

Example 2: Developer with Overtime Input:

Employee Type: 2 Name: Bob

ID: 102

Salary: 40000

Extra Hours: 10

Output:

Employee: Bob (ID: 102) Role: Developer

Base Salary: 40000

Overtime Compensation: 5000

Total Earnings: 45000

Example 3: Invalid Employee Type Input:

Employee Type: 3

Output:

Invalid employee type.

SOLUTION:

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

// Base class Employee
class Employee {
protected:
    string name;
    int id;
    int salary;

public:
    // Constructor to initialize employee details
    Employee(string n, int i, int s) : name(n), id(i), salary(s) {}

    // Virtual function to calculate total earnings (to be overridden)
    virtual void calculateTotalEarnings() = 0;

    // Function to display basic details
    void displayDetails() {
        cout << "Employee: " << name << " (ID: " << id << ")" << endl;
        cout << "Base Salary: " << salary << endl;
    }
};

// Derived class Manager
class Manager : public Employee {
private:
    int rating;  // Performance rating (1 to 5)

public:
    // Constructor to initialize Manager details
    Manager(string n, int i, int s, int r) : Employee(n, i, s), rating(r) {}

    // Override the function to calculate total earnings for a Manager
    void calculateTotalEarnings() override {
        // Bonus calculation: 10% of salary per rating point
        double bonus = (rating * 0.1) * salary;
        double totalEarnings = salary + bonus;

        // Display the manager's details and total earnings
        displayDetails();
        cout << "Role: Manager" << endl;
        cout << "Bonus: " << bonus << endl;
        cout << "Total Earnings: " << totalEarnings << endl;
    }
};

```

```cpp
// Derived class Developer
class Developer : public Employee {
private:
    int extraHours;  // Extra hours worked by the developer

public:
    // Constructor to initialize Developer details
    Developer(string n, int i, int s, int h) : Employee(n, i, s), extraHours(h) {}

    // Override the function to calculate total earnings for a Developer
    void calculateTotalEarnings() override {
        // Overtime compensation: $500 per hour
        double overtimeCompensation = extraHours * 500;
        double totalEarnings = salary + overtimeCompensation;

        // Display the developer's details and total earnings
        displayDetails();
        cout << "Role: Developer" << endl;
        cout << "Overtime Compensation: " << overtimeCompensation << endl;
        cout << "Total Earnings: " << totalEarnings << endl;
    }
};

int main() {
    int employeeType;
    string name;
    int id, salary, rating, extraHours;

    // Input: Employee type (1 for Manager, 2 for Developer)
    cout << "Enter employee type (1 for Manager, 2 for Developer): ";
    cin >> employeeType;

    if (employeeType == 1) {  // Manager
        cout << "Enter name: ";
        cin.ignore();  // To clear the input buffer
        getline(cin, name);
        cout << "Enter ID: ";
        cin >> id;
        cout << "Enter salary: ";
        cin >> salary;
        cout << "Enter performance rating (1-5): ";
        cin >> rating;

        // Create Manager object and calculate total earnings
        Manager manager(name, id, salary, rating);
        manager.calculateTotalEarnings();
```

```cpp
};

int main() {
    int employeeType;
    string name;
    int id, salary, rating, extraHours;

    // Input: Employee type (1 for Manager, 2 for Developer)
    cout << "Enter employee type (1 for Manager, 2 for Developer): ";
    cin >> employeeType;

    if (employeeType == 1) {  // Manager
        cout << "Enter name: ";
        cin.ignore();  // To clear the input buffer
        getline(cin, name);
        cout << "Enter ID: ";
        cin >> id;
        cout << "Enter salary: ";
        cin >> salary;
        cout << "Enter performance rating (1-5): ";
        cin >> rating;

        // Create Manager object and calculate total earnings
        Manager manager(name, id, salary, rating);
        manager.calculateTotalEarnings();

    } else if (employeeType == 2) {  // Developer
        cout << "Enter name: ";
        cin.ignore();  // To clear the input buffer
        getline(cin, name);
        cout << "Enter ID: ";
        cin >> id;
        cout << "Enter salary: ";
        cin >> salary;
        cout << "Enter extra hours worked: ";
        cin >> extraHours;

        // Create Developer object and calculate total earnings
        Developer developer(name, id, salary, extraHours);
        developer.calculateTotalEarnings();

    } else {
        cout << "Invalid employee type." << endl;  // Invalid employee type
    }

    return 0;
}
```