

Day-1 (Basic of C++)

Name:- Kuldeep

UID:- 22BCS10071

Section:- KPIT 901/A

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:

$$\text{Sum} = n \times (n+1) / 2 .$$

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

- $1 \leq n \leq 10^4$.

Output Format

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

Test Cases:

Example 1

Input:

5

Output:

15

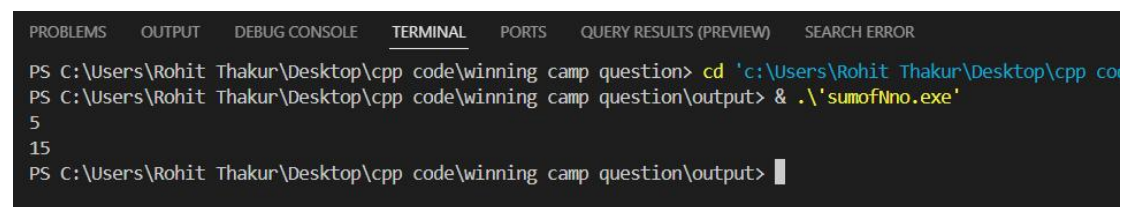
Explanation:

Using the formula, $\text{Sum} = 5 \times (5+1) / 2 = 15$.

Code:-

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

int main() {
    int n;
    cin >> n;
    int sum = n * (n + 1) / 2;
    cout << sum << endl;
    return 0;
}
```

Output:-

```
PS C:\Users\Rohit Thakur\Desktop\cpp code\winning camp question> cd 'c:\Users\Rohit Thakur\Desktop\cpp co
PS C:\Users\Rohit Thakur\Desktop\cpp code\winning camp question\output> & .\'sumofNno.exe'
5
15
PS C:\Users\Rohit Thakur\Desktop\cpp code\winning camp question\output> |
```

Easy:**1)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.

Input Format

One integer n.

Constraints

- $1 \leq n \leq 10^9$

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.

CODE:-

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

int main() {
    int n;
    cin >> n;
    int count = 0;

    while (n > 0) {
        n /= 10;
        count++;
    }

    cout << count << endl;
    return 0;
}
```

Output:-

```
PS C:\Users\Rohit Thakur\Desktop\cpp code\winning camp question\output> cd 'c:\Users\Rohit Thakur\Desktop\cpp c
PS C:\Users\Rohit Thakur\Desktop\cpp code\winning camp question\output> & .\'countdigit_in_no.exe'
12345
5
PS C:\Users\Rohit Thakur\Desktop\cpp code\winning camp question\output> █
```

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

$$1 \leq \text{radius, length, breadth, base, height} \leq 10^3$$

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 $3.14159 * 5^2 = 78.53975$.
- 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 $0.5 * 3 * 7 = 10.5$.

Code:-

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

const double PI = 3.14159;

double area(double radius) {
    return PI * radius * radius;
}
```

```

double area(double length, double breadth) {
    return length * breadth;
}

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

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

    cin >> radius;
    cout << area(radius) << endl;

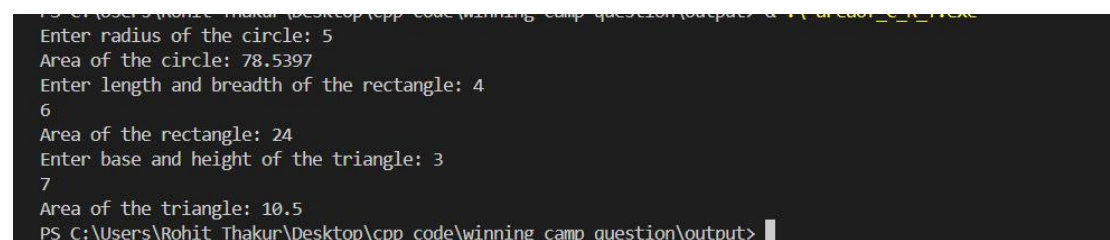
    cin >> length >> breadth;
    cout << area(length, breadth) << endl;

    cin >> base >> height;
    cout << area(base, height) << endl;

    return 0;
}

```

Output:-



```

PS C:\Users\Rohit Thakur\Desktop\cpp code\winning camp question\output> 5
Area of the circle: 78.5397
Enter length and breadth of the rectangle: 4
6
Area of the rectangle: 24
Enter base and height of the triangle: 3
7
Area of the triangle: 10.5
PS C:\Users\Rohit Thakur\Desktop\cpp code\winning camp question\output>

```

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.

Code:-

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

```
class Account {
public:
```

```

    virtual void calculateInterest() = 0; // Pure virtual function
};

class SavingsAccount : public Account {
private:
    double balance;
    double rate;
    int time;

public:
    SavingsAccount(double b, double r, int t) : balance(b), rate(r), time(t) {}

    void calculateInterest() override {
        double interest = balance * (rate / 100) * time;
        cout << "Savings Account Interest: " << interest << endl;
    }
};

class CurrentAccount : public Account {
private:
    double balance;
    double maintenanceFee;

public:
    CurrentAccount(double b, double fee) : balance(b), maintenanceFee(fee) {}

    void calculateInterest() override {
        balance -= maintenanceFee;
        cout << "Balance after fee deduction: " << balance << endl;
    }
};

int main() {
    int accountType;
    cout << "Account Type (1 for Savings, 2 for Current): ";
    cin >> accountType;

    if (accountType == 1) {
        double balance, rate;
        int time;
        cout << "Balance: ";
        cin >> balance;
        cout << "Interest Rate: ";
        cin >> rate;
        cout << "Time: ";
        cin >> time;
    }
}

```

```

        SavingsAccount savings(balance, rate, time);
        savings.calculateInterest();
    } else if (accountType == 2) {
        double balance, fee;
        cout << "Balance: ";
        cin >> balance;
        cout << "Maintenance Fee: ";
        cin >> fee;

        CurrentAccount current(balance, fee);
        current.calculateInterest();
    } else {
        cout << "Invalid account type." << endl;
    }

    return 0;
}

```

Output:-

```

PS C:\Users\Rohit Thakur\Desktop\cpp code\winning camp question\output> & .\'polymorphism_banking_trans
Account Type (1 for Savings, 2 for Current): 1
Balance: 10000
Interest Rate: 5
Time: 3
Savings Account Interest: 1500

```

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.

CODE:-

```

#include <iostream>
#include <string>

class Employee {
protected:
    std::string name;
    int id;
    int salary;

public:
    Employee(std::string n, int i, int s) : name(n), id(i), salary(s) {}
    virtual void calculateEarnings() = 0;
    virtual void display() = 0;
};

class Manager : public Employee {
private:
    int rating;

public:
    Manager(std::string n, int i, int s, int r) : Employee(n, i, s), rating(r) {}

    void calculateEarnings() override {
        double bonus = 0.1 * salary * rating;
        double totalEarnings = salary + bonus;
        std::cout << "Employee: " << name << " (ID: " << id << ")\n"
            << "Role: Manager\n"
            << "Base Salary: " << salary << "\n"
            << "Bonus: " << bonus << "\n"
            << "Total Earnings: " << totalEarnings << "\n";
    }

    void display() override {
        calculateEarnings();
    }
};

class Developer : public Employee {
private:
    int extraHours;

public:
    Developer(std::string n, int i, int s, int hours) : Employee(n, i, s), extraHours(hours) {}

    void calculateEarnings() override {
        double overtimeCompensation = 500 * extraHours;
    }
};

```

```

        double totalEarnings = salary + overtimeCompensation;
        std::cout << "Employee: " << name << " (ID: " << id << ")\n"
            << "Role: Developer\n"
            << "Base Salary: " << salary << "\n"
            << "Overtime Compensation: " << overtimeCompensation << "\n"
            << "Total Earnings: " << totalEarnings << "\n";
    }

    void display() override {
        calculateEarnings();
    }
};

int main() {
    int type;
    std::cout << "Employee Type (1 for Manager, 2 for Developer): ";
    std::cin >> type;

    if (type < 1 || type > 2) {
        std::cout << "Invalid employee type.\n";
        return 0;
    }

    std::string name;
    int id, salary;

    std::cout << "Name: ";
    std::cin >> name;
    std::cout << "ID: ";
    std::cin >> id;
    std::cout << "Salary: ";
    std::cin >> salary;

    if (type == 1) {
        int rating;
        std::cout << "Rating (1-5): ";
        std::cin >> rating;
        Manager manager(name, id, salary, rating);
        manager.display();
    } else if (type == 2) {
        int extraHours;
        std::cout << "Extra Hours: ";
        std::cin >> extraHours;
        Developer developer(name, id, salary, extraHours);
        developer.display();
    }
}

```

```
    return 0;  
}
```

Output:-

```
PS C:\Users\Rohit Thakur\Desktop\cpp code\winning camp question> cd 'c:\Users\Rohit Thakur\Desktop\cpp code\winning camp question\out  
PS C:\Users\Rohit Thakur\Desktop\cpp code\winning camp question\output> & .\inheritance_employee_management_system.exe'  
Employee Type (1 for Manager, 2 for Developer): 1  
Name: Alice  
ID: 101  
Salary: 50000  
Rating (1-5): 4  
Employee: Alice (ID: 101)  
Role: Manager  
Base Salary: 50000  
Bonus: 20000  
Total Earnings: 70000  
PS C:\Users\Rohit Thakur\Desktop\cpp code\winning camp question\output> |
```