Name- Aman Bansal UID- 22BCS13365 Section/Group- KPIT-901/B

Day 1

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

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

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, 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:

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

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

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

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

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.

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:

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

int main() {
    int n;
    cin >> n;

    if (n == 0) {
        cout << 1 << endl;
        return 0;
    }

    int count = 0;
    n = abs(n); // Handle negative numbers

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

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

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` $\leq 10^3$
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.

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 * 10^2 = 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 * 1^2 = 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:

```
#include <iostream>
using namespace std;
#define PI 3.14159


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

```
int area(int length, int breadth) {
    return length * breadth;
}
double area(double base, double height, bool isTriangle) {
    return 0.5 * base * height;
}

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

    cin >> radius;
    cin >> length >> breadth;
    cin >> base >> height;

    cout.precision(5);
    cout << fixed;
    cout << area(radius) << endl;
    cout << area(length, breadth) << endl;
    cout << area(base, height, true) << endl;

    return 0;
}
```

```
5
7 4
6
15
78.53975
28
90


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

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

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**:

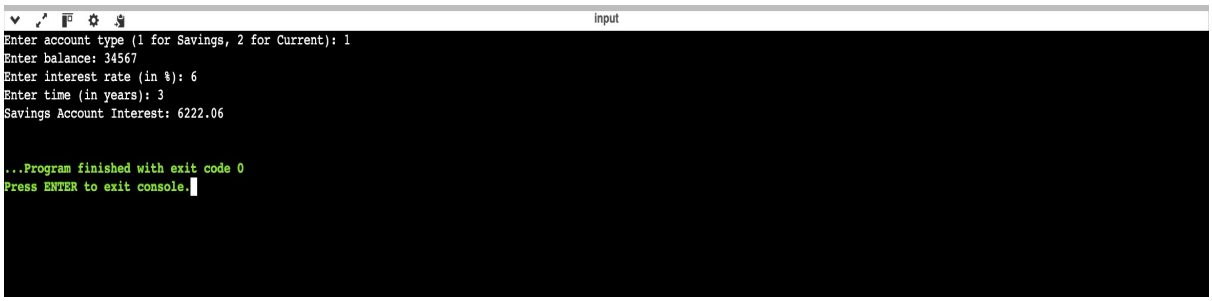
Invalid account type.

SOLUTION:

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

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

public:
    Account(int b) : balance(b) {}
    virtual void calculateInterest() = 0; // Pure virtual function
};

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

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

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

// Derived class: CurrentAccount
class CurrentAccount : public Account {
private:
    int maintenanceFee;

public:
    CurrentAccount(int b, int fee) : Account(b), maintenanceFee(fee) {}

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

```
};

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

    if (accountType == 1) {
        int balance, time;
        double rate;

        cout << "Enter Balance: ";
        cin >> balance;
        cout << "Enter Interest Rate (in percentage): ";
        cin >> rate;
        cout << "Enter Time (in years): ";
        cin >> time;

        if (balance < 1000 || balance > 1000000 || rate < 1 || rate > 15 || time < 1 || time > 10) {
            cout << "Invalid input.\n";
            return 1;
        }

        SavingsAccount savings(balance, rate, time);
        savings.calculateInterest();

    } else if (accountType == 2) {
        int balance, fee;

        cout << "Enter Balance: ";
        cin >> balance;
        cout << "Enter Monthly Maintenance Fee: ";
        cin >> fee;

        if (balance < 1000 || balance > 1000000 || fee < 50 || fee > 500) {
            cout << "Invalid input.\n";
            return 1;
        }

        CurrentAccount current(balance, fee);
        current.calculateInterest();

    } else {
        cout << "Invalid account type.\n";
    }
```

```
    return 0;
}
```



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

<u>Test Cases:</u>

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:

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

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

public:
    Employee(string n, int i, int s) : name(n), id(i), salary(s) {}

    virtual void display() {
        cout << "Employee: " << name << " (ID: " << id << ")\n";
        cout << "Base Salary: " << salary << "\n";
```

```
    }

    virtual int calculateTotalEarnings() = 0; // Pure virtual function
};

// Derived class: Manager
class Manager : public Employee {
private:
    int rating;

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

    int calculateBonus() {
        return (rating * 0.1 * salary);
    }

    int calculateTotalEarnings() override {
        return salary + calculateBonus();
    }

    void display() override {
        Employee::display();
        cout << "Role: Manager\n";
        cout << "Bonus: " << calculateBonus() << "\n";
        cout << "Total Earnings: " << calculateTotalEarnings() << "\n";
    }
};

// Derived class: Developer
class Developer : public Employee {
private:
    int extraHours;

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

    int calculateOvertime() {
        return extraHours * 500;
    }
```

```
    int calculateTotalEarnings() override {
        return salary + calculateOvertime();
    }

    void display() override {
        Employee::display();
        cout << "Role: Developer\n";
        cout << "Overtime Compensation: " << calculateOvertime() << "\n";
        cout << "Total Earnings: " << calculateTotalEarnings() << "\n";
    }
};

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

    if (employeeType == 1) {
        string name;
        int id, salary, rating;

        cout << "Enter Name: ";
        cin >> name;
        cout << "Enter ID: ";
        cin >> id;
        cout << "Enter Salary: ";
        cin >> salary;
        cout << "Enter Performance Rating (1-5): ";
        cin >> rating;

        if (rating < 1 || rating > 5 || salary < 10000 || salary > 1000000) {
            cout << "Invalid input.\n";
            return 1;
        }

        Manager manager(name, id, salary, rating);
        manager.display();

    } else if (employeeType == 2) {
        string name;
        int id, salary, extraHours;
```

```
        cout << "Enter Name: ";
        cin >> name;
        cout << "Enter ID: ";
        cin >> id;
        cout << "Enter Salary: ";
        cin >> salary;
        cout << "Enter Extra Hours Worked: ";
        cin >> extraHours;

        if (extraHours < 0 || extraHours > 100 || salary < 10000 || salary > 1000000) {
            cout << "Invalid input.\n";
            return 1;
        }

        Developer developer(name, id, salary, extraHours);
        developer.display();

    } else {
        cout << "Invalid employee type.\n";
    }
    return 0;
}
```