

## **Day 1**

Name: Sumer Singh

UID: 22BCS12171

Section: 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:

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

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:**

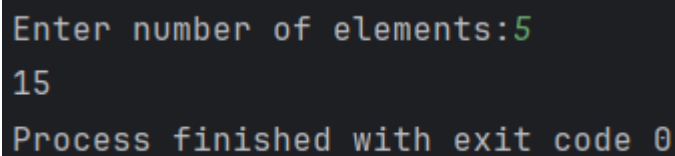
Using the formula,  $\text{Sum} = 5 \times (5+1) / 2 = 15$  .

**Code:**

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cout<<"Enter number of elements:";
    cin>>n;

    int sum = n*(n+1)/2;
    cout<<sum;
}
```

Output:

A screenshot of a terminal window showing the output of the C++ program. The text is as follows:  
Enter number of elements:5  
15  
Process finished with exit code 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.

## Output Format

Print the number of digits in n.

### Test Cases

#### Example 1:

##### Input:

12345

##### Output:

5

##### Explanation:

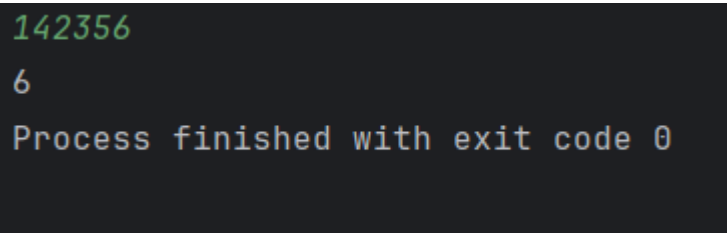
The number 12345 has 5 digits: 1, 2, 3, 4, 5.

##### Code:

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

int main()
{
    int n,a;
    cin>>n;
    int count=0;
    while(n!=0)
    {
        n=n/10;
        count++;
    }
    cout<<count;
}
```

##### Output:



```
12345
5
Process finished with exit code 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**

$1 \leq \text{radius, length, breadth, base, height} \leq 10^3$   
Use 3.14159 for the value of  $\pi$ .

#### **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;
class Circle
{
```

```
public:
    void Area(int r)
    {
        cout<<"Area of circle is: "<<3.14*r*r;
    }
};
```

```
class Rectangle: public Circle
```

```
{
public:
    void Area(int l,int b)
    {
        cout<<"Area of rectangle is:"<<l*b;
    }
};
```

```
int main()
{
    int l=10,b=10,r=10;
    Rectangle re;
    re.Area(l,b);
}
```

Output:

```
Area of rectangle is:100
Process finished with exit code 0
```

## **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:**  $\text{Interest} = \text{Balance} \times \text{Rate} \times \text{Time}$ .
- **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

### Code:

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

class Account {
protected:
    double balance;
public:
    Account(double bal) : balance(bal) {}
    virtual void calculateInterest() = 0;
    virtual ~Account() {}
};
```

```

class SavingsAccount : public Account {
    double rate;
    int time;
public:
    SavingsAccount(double bal, double r, int t) : Account(bal), rate(r), time(t) {}
    void calculateInterest() override {
        double interest = balance * (rate / 100) * time;
        cout << fixed << setprecision(2) << "Interest: " << interest << endl;
    }
};

class CurrentAccount : public Account {
    double maintenanceFee;
public:
    CurrentAccount(double bal, double fee) : Account(bal), maintenanceFee(fee) {}
    void calculateInterest() override {
        balance -= maintenanceFee;
        cout << fixed << setprecision(2) << "Remaining Balance after Maintenance Fee: " << balance
        << endl;
    }
};

int main() {
    int accountType;
    cout<<"1: for Savings Account\n2: for Current Account";
    cin >> accountType;

    if (accountType == 1) {
        double balance, rate;
        int time;
        cout<<"Balance,Rate,Time";
        cin >> balance >> rate >> time;
        SavingsAccount sa(balance, rate, time);
        sa.calculateInterest();
    } else if (accountType == 2) {
        double balance, maintenanceFee;

```

```

        cout<<"Balance,Maintenance Fee";
        cin >> balance >> maintenanceFee;
        CurrentAccount ca(balance, maintenanceFee);
        ca.calculateInterest();
    }
}

```

Output:

```

1: for Savings Account
2: for Current Account
1
Balance,Rate,Time
500000
5
10
Interest: 250000.00

Process finished with exit code 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:  $1 \leq \text{type} \leq 2$  .



- 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

##### **Code:**

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

class Employee {
protected:
    string name;
    int id;
    double salary;
public:
    Employee(string n, int i, double s) : name(n), id(i), salary(s) {}
    virtual void calculateEarnings() = 0;
    virtual ~Employee() {}
};

class Manager : public Employee {
    int rating;
public:
```

```

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

void calculateEarnings() override {
    double bonus = salary * 0.1 * rating;
    double totalEarnings = salary + bonus;
    cout << "Employee: " << name << " (ID: " << id << ")\n";
    cout << "Role: Manager\n";
    cout << "Base Salary: " << fixed << setprecision(2) << salary << "\n";
    cout << "Bonus: " << bonus << "\n";
    cout << "Total Earnings: " << totalEarnings << "\n";
}
};

class Developer : public Employee {
    int extraHours;
public:
    Developer(string n, int i, double s, int h) : Employee(n, i, s), extraHours(h) {}
    void calculateEarnings() override {
        double overtimeCompensation = extraHours * 500;
        double totalEarnings = salary + overtimeCompensation;
        cout << "Employee: " << name << " (ID: " << id << ")\n";
        cout << "Role: Developer\n";
        cout << "Base Salary: " << fixed << setprecision(2) << salary << "\n";
        cout << "Overtime Compensation: " << overtimeCompensation << "\n";
        cout << "Total Earnings: " << totalEarnings << "\n";
    }
};

int main() {
    int employeeType;
    cout<<"1: Manager\n2: Developer";
    cin >> employeeType;

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

```

```

        double salary;
        int rating;
        cout<<"Name,ID,Salary,Rating";
        cin >> name >> id >> salary >> rating;
        Manager m(name, id, salary, rating);
        m.calculateEarnings();
    } else if (employeeType == 2) {
        string name;
        int id;
        double salary;
        int extraHours;
        cout<<"Name,ID,Salary,Extra Hours";
        cin >> name >> id >> salary >> extraHours;
        Developer d(name, id, salary, extraHours);
        d.calculateEarnings();
    }
}

```

Output:

```

1: Manager
2: Developer
1
Name,ID,Salary,Rating
Sumer 1 1500000 10
Employee: Sumer (ID: 1)
Role: Manager
Base Salary: 1500000.00
Bonus: 1500000.00
Total Earnings: 3000000.00

Process finished with exit code 0

```