ASSIGNMENT

RAJIV KUMAR

22BCS16830

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

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

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 .

CODE:

```
#include<iostream>

using namespace std;

void summation(int n){

   int sum;

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

   cout<<"sum of natural number:"<<sum<<endl;

}

int main(){

    int n;

    cout<<"input the range to find sum of natural number."<<endl;

    cin>>n;

    summation(n);

}
```

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.

Input Format

One integer n.

Constraints

- $2 \leq n \leq 10^5$

Output Format

Print "Prime" if n is prime, otherwise print "Not Prime".

Test Cases:

Example 1

Input:

```
7
```

Output:

```
Prime
```

Explanation:
7 has no divisors other than 1 and itself, so it is a prime number.

Example 2

Input:

```
9
```

Output:

```
Not Prime
```

Explanation:
9 is divisible by 3, so it is not a prime number.

Example 3

Input:

```
2
```

Output:

```
Prime
```

Explanation:
2 is a prime number as it has only two divisors: 1 and 2.

Code:

```
#include<iostream>

using namespace std;

void primeno(int n,int &cnt){




  for(int i=1;i<=n;i++){

     if(n%i==0)

     {

        cnt++;

     }

  }


}

int main()

{


   int n;

   int cnt=0;

   cin>>n;

   primeno(n,cnt);

   if(cnt==2)
```

```
    {
        cout<<"the number is prime no"<<endl;
    }
    else
    cout<<"the no.is not prime"<<endl;
    return 0;
}
```

3) Print Odd Numbers up to N

Objective

Print all odd numbers between 1 and n, inclusive. Odd numbers are integers that are not divisible by 2. These numbers should be printed in ascending order, separated by spaces.

This problem is a simple introduction to loops and conditional checks. The goal is to use a loop to iterate over the numbers and check if they are odd using the condition i%2≠0.

Task

Given an integer n, print all odd numbers from 1 to n, inclusive.

Input Format

One integer n, the upper limit of the range.

Constraints

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

Output Format

A single line containing all odd numbers from 1 to n, separated by spaces.

Test Cases:

Example 1

Input:

```
10
```

Output:

```
1 3 5 7 9
```

Example 2

Input:

```
7
```

Output:

```
1 3 5 7
```

Example 3

Input:

```
1
```

Output:

```
1
```

Code:

```
#include<iostream>
using namespace std;
void oddno(int n){
    for(int i=0;i<n;i++){
        if(i%2!=0){
            cout<<i<<" ";
        }
    }

}
int main(){
     int n;
     cout<<"input the range to find odd  number."<<endl;
     cin>>n;
     oddno(n);
}
```

4) Sum of Odd Numbers up to N

Objective

Calculate the sum of all odd numbers from 1 to n. An odd number is an integer that is not divisible by 2.The sum of odd numbers, iterate through all the numbers from 1 to n, check if each number is odd, and accumulate the sum.

Task

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

Input Format

One integer n, the upper limit of the range.

Constraints

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

Output Format

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

Test Cases:

Example 1:
Input:

```
5
```

Output:

```
9
```

Explanation:
The odd numbers are 1, 3, 5. Their sum is 1+3+5=9.

Example 2:
Input:

```
10
```

Output:

```
25
```

Explanation:
The odd numbers are 1, 3, 5, 7, 9. Their sum is 1+3+5+7+9=25.

Example 3:
Input:

```
1
```

Output:

```
1
```

Explanation:
The only odd number is 1.

Code:

```
#include<iostream>

using namespace std;

void oddno(int n){

    int sum=0;

   for(int i=0;i<n;i++){


       if(i%2!=0){

          sum+=i;

       }


   }

   cout<<"sum of odd number upto n :"<<sum;


}
```

```
int main(){
    int n;
    cout<<"input the range to find odd  number."<<endl;
    cin>>n;
    oddno(n);
}
```

5) 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 $10\times n$.

Input Format

One integer n.

Constraints

- $1 \leq n \leq 100$

Output Format

For each integer i from 1 to 10, print the product $n\times i$ in the format:
$n\times i=product$.

Test Cases

Example 1:
Input:

3

Output:

Copy code

3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30

Example 2:

Input:

7

Output:

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

Example 3:

Input:

10

Output:

10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100

Code:

```
#include<iostream>
using namespace std;
void table(int n){

   for(int i=1;i<=10;i++){

      cout<<n*i<<" ";
      }

   }


int main(){
    int n;
    cout<<"input the table number"<<endl;
    cin>>n;
    table(n);
}
```

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.

Code:

```
#include<iostream>
using namespace std;
void countdigits(int n){
   int count=0;

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

       }
       cout<<"the number of digits in n:"<<count<<endl;

   }


int main(){
    int n;
    cout<<"input the  number"<<endl;
    cin>>n;
    countdigits(n);
}
```

2) 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.

Input Format

One integer n.

Constraints

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

Output Format

Print the number with its digits in reverse order.

Test Cases
Example 1:
Input:
12345

Output:

54321

Explanation:
The digits of 12345 in reverse order are 54321.

Example 2:
Input:

9876

Output:

6789

Explanation:
The digits of 9876 in reverse order are 6789.

Code:

```
#include<iostream>

using namespace std;

void reverse(int n){

    int rev=0;

    int digits;


        while(n>0){

           digits=n%10;

           rev=rev*10+digits;

           n=n/10;


        }

        cout<<"the reverse number :"<<rev<<endl;


    }



int main(){

     int n;

     cout<<"input the  number"<<endl;

     cin>>n;
```

```
    reverse(n);

}
```

Example 3:
Input:

1000

Output:

1

Explanation:
The digits of 1000 in reverse order are 0001, which is equivalent to 1.

3) 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.

Input Format

One integer n.

Constraints

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

Output Format

Print the largest digit in the number n.

Test Cases
Example 1:
Input:
2734

Output:

7

Explanation:
The digits of 2734 are 2, 7, 3, and 4. The largest digit is 7.

Example 2:
Input:

9450

Output:

9

Explanation:
The digits of 9450 are 9, 4, 5, and 0. The largest digit is 9.

Example 3:
Input:

1111

Output:

1

Explanation:
All digits of 1111 are 1, so the largest digit is also 1.

4) Check if a Number is a Palindrome

Objective

Check whether a given number is a palindrome or not. A number is called a palindrome if it reads the same backward as forward. For example, 121 is a palindrome because reading it from left to right is the same as reading it from right to left. Similarly, 12321 is also a palindrome, but 12345 is not.

Task

Given an integer n, print "Palindrome" if the number is a palindrome, otherwise print "Not Palindrome".

Input Format

One integer n.

Constraints

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

Output Format

Print "Palindrome" if the number is a palindrome, otherwise print "Not Palindrome".

Explanation

For input n=121n = 121n=121, the number is the same when reversed, so it is a palindrome.
For input n=12345n = 12345n=12345, the number is not the same when reversed, so it is not a palindrome.

Test Cases

Example 1

Input:

121

Output:

Palindrome

Explanation:
The number 121 reads the same backward as forward.

Example 2:
Input:

12345

Output:

Not Palindrome

Explanation:
The number 12345 does not read the same backward as forward.

Example 3:
Input:

12321

Output:

Palindrome

Explanation:
The number 12321 reads the same backward as forward.

Code:

```
#include <iostream>

using namespace std;



int main()

{

   int i,n,originalnum,rev=0,remainder;

   cout<<"enter the number to reversed"<<endl;

   cin>>n;

   originalnum=n;

   while(n>0){

      remainder=n%10;

      rev=rev*10+remainder;

      n=n/10;

   }

   cout<<"the reversed number is:"<<rev<<endl;

   if(originalnum==rev){



   cout<<"palindrome number"<<endl;
```

```
    }
    else
    cout<<"the number is not palindrome"<<endl;
    return 0;
}
```

5) 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.

Input Format

One integer n.

Constraints

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

Output Format

Print the sum of the digits of n.

Test Cases

Example 1

Input:

12345

Output:

15

Explanation:
The digits of 12345 are 1, 2, 3, 4, and 5. The sum is 1 + 2 + 3 + 4 + 5 = 15.

Example 2:
Input:

4567

Output:

22

Explanation:
The digits of 4567 are 4, 5, 6, and 7. The sum is 4 + 5 + 6 + 7 = 22.

Example 3:
Input:

999

Output:

27

Explanation:
The digits of 999 are 9, 9, and 9. The sum is 9 + 9 + 9 = 27.

Code:

```
#include <iostream>

using namespace std;



int main() {

    int n = 12345;  // The number whose digits we want to sum

    int sum = 0;    // Variable to store the sum of the digits
```

```
    // Loop to extract and sum digits
    while (n > 0) {
        sum += n % 10;  // Add the last digit to sum
        n = n / 10;     // Remove the last digit
    }

    // Output the result
    cout << "Sum of the digits: " << sum << 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.

2) Function Overloading with Hierarchical Structure.
Objective
Write a program that demonstrates function overloading to calculate the salary of employees at different levels in a company hierarchy. Implement overloaded functions to compute salary for:

- Intern (basic stipend).
- Regular employee (base salary + bonuses).
- Manager (base salary + bonuses + performance incentives).

Input Format:
The program should accept:

1. Intern: Basic stipend.
2. Regular employee: Base salary and bonuses.
3. Manager: Base salary, bonuses, and performance incentives.

Constraints

- $1 \leq stipend, base\ salary, bonuses, incentives \leq 10^6$.

Output Format
Print the calculated salary for each level of the hierarchy on a new line.
Test Cases:
Example 1
Input:
Stipend = 10000
base salary = 50000, bonuses = 20000
base salary = 80000, bonuses = 30000, incentives = 20000
Output:
Intern Salary: 10000
Employee Salary: 70000
Manager Salary: 130000
Explanation:

- Intern receives only the stipend: 10000.

- Regular employee salary is calculated as 50000+20000=70000.
- Manager's salary includes all components: 80000+30000+20000=

Example 2
Input:
Stipend = 15000
base salary = 60000, bonuses = 25000
base salary = 100000, bonuses = 40000, incentives = 30000
Output:
Intern Salary: 15000
Employee Salary: 85000
Manager Salary: 170000

Example 3
Input:
stipend = 5000
base_salary = 45000, bonuses = 15000
base_salary = 70000, bonuses = 20000, incentives = 10000
Output:
Intern Salary: 5000
Employee Salary: 60000
Manager Salary: 100000

3) **Encapsulation with Employee Details**
Objective
Write a program that demonstrates encapsulation by creating a class Employee. The class should have private attributes to store:
Employee ID.
Employee Name.
Employee Salary.
Provide public methods to set and get these attributes, and a method to display all details of the employee.

Input *Format*
The program should accept:

1. Employee ID as an integer.
2. Employee Name as a string.
3. Employee Salary as a floating-point number.

Constraints

- $1 \leq$ Employee ID $\leq 10^6$.
- Name can have up to 50 characters.
- $1.0 \leq$ Salary $\leq 10^7$.

Output Format
Print the employee details, including ID, Name, and Salary, on separate lines.
Test Cases:
Example 1
Input:
ID = 101
Name = John Doe
Salary = 75000.5
Output:
Employee ID: 101
Employee Name: John Doe
Employee Salary: 75000.5
Explanation:
Encapsulation ensures that employee details are accessed and modified only through public methods.
Example 2
Input:
ID = 202
Name = Jane Smith
Salary = 85000.75
Output:
Employee ID: 202
Employee Name: Jane Smith
Employee Salary: 85000.75

Example 3
Input:
ID = 303
Name = Robert Brown
Salary = 67000.0
Output:
Employee ID: 303
Employee Name: Robert Brown
Employee Salary: 67000.0

4) Inheritance with Student and Result Classes.

Objective
Create a program that demonstrates inheritance by defining:

- A base class Student to store details like Roll Number and Name.
- A derived class Result to store marks for three subjects and calculate the total and percentage.

Input Format
The program should accept:

1. Roll Number as an integer.

2. Name as a string.
3. Marks in three subjects as integers.

Constraints

- `1 ≤ Roll Number ≤ 10` [6.]
- Name can have up to 50 characters.
- `0 ≤ Marks in each subject ≤ 100.`

Output Format
Print the student details, marks, total, and percentage.

Test Cases:
Example 1
Input:
Roll Number = 101
Name = Alice Smith
Marks = 85, 90, 80
Output:
Roll Number: 101
Name: Alice Smith
Marks: 85, 90, 80
Total: 255
Percentage: 85.0%

Explanation:
The Result class inherits Student details and calculates Total=85 + 90 + 80 and Percentage = (255/300) × 100.

Example 2
Input:
Roll Number = 202
Name = Bob Martin
Marks = 70, 75, 65
Output:
Roll Number: 202
Name: Bob Martin
Marks: 70, 75, 65
Total: 210
Percentage: 70.0%
Example 3
Input:
Roll Number = 303
Name = Clara Brown
Marks = 95, 92, 88
Output:

Roll Number: 303
Name: Clara Brown
Marks: 95, 92, 88
Total: 275
Percentage: 91.67%

5) 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.

Constraints

- $1 \leq \texttt{Radius, Length, Breadth, Base, Height} \leq 10^4$.
- Use π=3.14159 for circle area calculations.

Output Format
Print the area of each shape on separate lines, rounded to 2 decimal places.
Test Cases:
Example 1
Input:
Radius = 7
Length = 10, Breadth = 5
Base = 8, Height = 6
Output:
Circle Area: 153.94
Rectangle Area: 50.00
Triangle Area: 24.00
Explanation:

- Circle area is calculated as $\pi r^2 = 3.14159 \times 7^2 = 153.94$.
- Rectangle area is Length × Breadth=10×5=50
- Triangle area is 0.5 × Base × Height=0.5×8×6=24 .

Example 2*:*
Input:
Radius = 5
Length = 12, Breadth = 8
Base = 15, Height = 10

Output:
Circle Area: 78.54
Rectangle Area: 96.00
Triangle Area: 75.00

Example 3:
Input:
Radius = 10
Length = 20, Breadth = 15
Base = 10, Height = 12
Output:
Circle Area: 314.16
Rectangle Area: 300.00
Triangle Area: 60.00

Hard:

1)Implementing Polymorphism for Shape Hierarchies.

Objective

Write a program to demonstrate runtime polymorphism in C++ using a base class Shape and derived classes Circle, Rectangle, and Triangle. The program should use virtual functions to calculate and print the area of each shape based on user input.

Input Format
The program should accept:

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

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 on a new line.
Test Cases:
Example 1
Input:
Radius = 5
Length = 4, breadth = 6
Base = 3, height = 7
Output:
Area of Circle: 78.53975

Area of Rectangle: 24
Area of Triangle: 10.5
Explanation:

- The area of the circle is 3.14159×52=78.53975.
- The area of the rectangle is 4×6=24.
- The area of the triangle is 0.5×3×7=10.5.

Example 2
Input:
Radius = 10
Length = 15, breadth = 8
Base = 12, height = 9
Output:
Area of Circle: 314.159
Area of Rectangle: 120
Area of Triangle: 54
Explanation:

- The area of the circle is $3.14159 \times 10^2 = 314.159$.
- The area of the rectangle is 15×8=120.
- The area of the triangle is 0.5×12×9=54.

Example 2
Input:
Radius = 1
Length = 2, breadth = 3
Base = 5, height = 8
Output:
Area of Circle: 3.14159
Area of Rectangle: 6
Area of Triangle: 20
Explanation:

- The area of the circle is 3.14159×1=3.14159.
- The area of the rectangle is 2×3=6.
- The area of the triangle is 0.5×5×8=20.

2) 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.

Constraints

- $1 \leq m, n, p \leq 10$.
- Matrix elements are integers in the range -100 to 100.

Output Format

- For **Matrix Addition**, print the resulting matrix.
- For **Matrix Multiplication**, print the resulting matrix.
- If dimensions are invalid for the chosen operation, output: "Invalid dimensions for operation."

Test Cases:

Example 1: Matrix Addition
Input:
Matrix A:
1 2
3 4
Matrix B:
5 6
7 8
Operation: 1
Output:
6 8
10 12

Example 2: Matrix Multiplication
Input:
Matrix A:
1 2 3
4 5 6
Matrix B:

7 8
9 10
11 12
Operation: 2
Output:
58 64
139 154

Example 3: Invalid Dimensions
Input:
Matrix A:
1 2 3
4 5 6
Matrix B:
7 8 9
10 11 12
Operation: 2
Output:
rust
Invalid dimensions for operation.

3) Polymorphism in Shape Classes

Objective:

Design a C++ program using polymorphism to calculate the area of different shapes:
A **Rectangle** (Area = Length × Breadth).
A **Circle** (Area = π × Radius²).
A **Triangle** (Area = ½ × Base × Height).
Create a base class Shape with a pure virtual function getArea(). Use derived classes Rectangle, Circle, and Triangle to override this function.

Input Format

- Shape type (1 for Rectangle, 2 for Circle, 3 for Triangle).
- For Rectangle: Length and Breadth.
- For Circle: Radius.
- For Triangle: Base and Height.

Constraints

- `Shape type is an integer: 1 ≤ type ≤ 3.`
- `Dimensions are integers: 1 ≤ Dimension Value ≤ 100.`

- Use π = 3.14159 for Circle Area calculation.

Test Cases:

Example 1: Rectangle Area

Input:
Shape Type: 1
Length: 5
Breadth: 4
Output:
The area of the rectangle is: 20

Example 2: Circle Area

Input:
Shape Type: 2
Radius: 7
Output:
The area of the circle is: 153.938

Example 3: Triangle Area

Input:
Shape Type: 3
Base: 10
Height: 6
Output:
The area of the triangle is: 30

4) Implement Multiple Inheritance to Simulate a Library System

Objective

Create a C++ program using multiple inheritance to simulate a library system. Design two base classes:

- Book to store book details (title, author, and ISBN).
- Borrower to store borrower details (name, ID, and borrowed book).

Create a derived class Library that inherits from both Book and Borrower. Use this class to track the borrowing and returning of books.

Input Format

- Book Details: Title, Author, ISBN.
- Borrower Details: Name, ID.
- Action type:
  - 1 to Borrow a Book.
  - 2 to Return a Book.

Constraints

- Book title and author are strings (max length 50).
- ISBN is an integer: 1000 ≤ ISBN ≤ 9999.
- Borrower ID is an integer: 1≤ID≤1000.

Test Cases:

Example 1: Rectangle Area
Input:
Book Details:
Title: C++ Basics
Author: John Doe
ISBN: 1234
Borrower Details:
Name: Alice
ID: 42
Output:
Borrower Alice (ID: 42) has borrowed "C++ Basics" by John Doe (ISBN: 1234).

Example 2: Return a Book
Input:
Book Details:
Title: C++ Basics
Author: John Doe
ISBN: 1234
Borrower Details:
Name: Alice
ID: 42
Action: 2
Output:
Borrower Alice (ID: 42) has returned "C++ Basics" by John Doe (ISBN: 1234).

Example 3: Invalid Action
Input:
Book Details:
Title: Advanced C++
Author: Jane Smith
ISBN: 5678
Borrower Details:
Name: Bob
ID: 99
Action: 3
Output:
Invalid action type.

5) 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 ≤ type ≤ 2 .`
- `Balance: 1000 ≤ balance ≤ 1,000,000 .`
- `Interest Rate: 1 ≤ rate ≤ 15.`
- `Time: 1 ≤ time ≤ 10 .`
- `Maintenance Fee: 50 ≤ fee ≤ 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.

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: $1 \leq type \leq 2$.
- Salary: $10{,}000 \leq salary \leq 1{,}000{,}000$.
- Rating: $1 \leq rating \leq 5$.
- Extra hours: $0 \leq hours \leq 100$.
- 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.

2) Multi-Level Inheritance for Vehicle Simulation
Objective
Create a C++ program to simulate a vehicle hierarchy using multi-level inheritance. Design a base class Vehicle that stores basic details (brand, model, and mileage). Extend it into the Car class to add attributes like fuel efficiency and speed. Further extend it into ElectricCar to include battery capacity and charging time. Implement methods to calculate:
Fuel Efficiency: Miles per gallon (for Car).
Range: Total distance the electric car can travel with a full charge.

Input Format

1. Vehicle Type (1 for Car, 2 for Electric Car).
2. Brand (string), Model (string), and Mileage (double).
3. For Car: Fuel (gallons) and Distance Covered (miles).
4. For Electric Car: Battery Capacity (kWh), Efficiency (miles per kWh).

Constraints

- `Vehicle type: 1 ≤ type ≤ 2.`
- `Mileage: 0 ≤ mileage ≤ 500,000.`
- `Fuel: 1 ≤ fuel ≤100.`
- `Distance: 1 ≤ distance ≤ 1,000.`
- `Battery Capacity: 10 ≤ capacity ≤ 150.`
- `Efficiency: 1 ≤ efficiency ≤ 10.`

Test Cases:

Example 1: Fuel Efficiency for Car

Input:
Vehicle Type: 1
Brand: Toyota
Model: Corolla
Mileage: 30000
Fuel: 15
Distance: 300

Output:
Vehicle: Toyota Corolla
Mileage: 30000
Fuel Efficiency: 20 miles/gallon

Example 2: Range of Electric Car

Input:
Vehicle Type: 2
Brand: Tesla
Model: Model S
Mileage: 5000
Battery Capacity: 100
Efficiency: 4

Output:
Vehicle: Tesla Model S
Mileage: 5000
Range: 400 miles

Example 3: Invalid Vehicle Type

Input:
Vehicle Type: 3

Output:
Invalid vehicle type.

3) Function Overloading for Complex Number Operations.

Objective

Design a C++ program using **function overloading** to perform arithmetic operations on complex numbers. Define a Complex class with real and imaginary parts. Overload functions to handle the following operations:

Addition: Sum of two complex numbers.

Multiplication: Product of two complex numbers.

Magnitude: Calculate the magnitude of a single complex number.

The program should allow the user to select an operation, input complex numbers, and display results in the format a + bi or a - bi (where b is the imaginary part).

Input Format

1. Operation Type (1 for Addition, 2 for Multiplication, 3 for Magnitude).

2. For Addition and Multiplication: Two complex numbers (real1, imaginary1, real2, imaginary2).
3. For Magnitude: One complex number (real, imaginary).

Constraints

- Operation Type: $1 \leq type \leq 3$.
- Real and Imaginary parts: $-10^3 \leq real, imaginary \leq 10^3$.

Test Cases:

Example 1: Addition of Complex Numbers
Input:
Operation Type: 1
Complex Number 1: 3 2
Complex Number 2: 1 -4
Output:
Result: 4 - 2i
Example 2: Multiplication of Complex Numbers
Input:
Operation Type: 2
Complex Number 1: 2 3
Complex Number 2: 1 -1
Output:
Result: 5 + 1i
Example 3: Magnitude of a Complex Number
Input:
Operation Type: 3
Complex Number: 3 4
Output:
Result: Magnitude = 5

Example 4: Invalid Operation
Input:
Operation Type: 4
Output:
Invalid operation type.
4) Polymorphism for Shape Area Calculations
Objective

Create a C++ program that uses polymorphism to calculate the area of various shapes. Define a base class Shape with a virtual method calculateArea(). Extend this base class into the following derived classes:

Rectangle: Calculates the area based on length and width.

Circle: Calculates the area based on the radius.

Triangle: Calculates the area using base and height.

The program should use dynamic polymorphism to handle these shapes and display the area of each.

Input Format

1. Shape Type (1 for Rectangle, 2 for Circle, 3 for Triangle).
2. For Rectangle: Length and Width (float).
3. For Circle: Radius (float).
4. For Triangle: Base and Height (float).

Constraints

- `Shape Type:` $1 \leq type \leq 3$.
- `Length, Width, Radius, Base, and Height:` $1.0 \leq value \leq 10^4$.
- Use π=3.14159 for calculations.

Test Cases:

Example 1: Rectangle Area
Input:
Shape Type: 1
Length: 5
Width: 10
Output:
Shape: Rectangle
Area: 50.00

Example 2: Circle Area
Input:
Shape Type: 2
Radius: 7
Output:
Shape: Circle
Area: 153.94

Example 3: Triangle Area
Input:
Shape Type: 3
Base: 8
Height: 6
Output:
Shape: Triangle

Area: 24.00
Example 4: Invalid Shape Type
Input:
Shape Type: 4
Output:
Invalid shape type.

5) Advanced Function Overloading for Geometric Shapes
Objective
Create a C++ program that demonstrates **function overloading** to calculate the area of different geometric shapes. Implement three overloaded functions named calculateArea that compute the area for the following shapes:
Circle: Accepts the radius.
Rectangle: Accepts the length and breadth.
Triangle: Accepts the base and height.
Additionally, use a menu-driven program to let the user choose the type of shape and input the respective parameters. Perform necessary validations on the input values.

Input Format

- An integer $1 \leq choice \leq 3$ representing the shape type:
    1. Circle
    2. Rectangle
    3. Triangle
- For each shape:
    - **Circle**: A positive floating-point number for the radius.
    - **Rectangle**: Two positive floating-point numbers for length and breadth.
    - **Triangle**: Two positive floating-point numbers for base and height.

Constraints

- $1 \leq choice \leq 3$.
- $0.0 < parameters \leq 10^6$.

Test Cases:
Example 1: Circle
Input:
Choice: 1
Radius: 7.5
Output:
Shape: Circle
Radius: 7.5

Area: 176.714
Example 2: Rectangle
Input:
Choice: 2
Length: 8.0
Breadth: 4.5
Output:
Shape: Rectangle
Length: 8.0
Breadth: 4.5
Area: 36.000
Example 3: Triangle
Input:
Choice: 3
Base: 6.0
Height: 9.0
Output:
Shape: Triangle
Base: 6.0
Height: 9.0
Area: 27.000
Example 4: Invalid Choice
Input:
Choice: 4
Output:
Invalid choice. Please select a valid shape type.