

## **Day1**

### **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,  $\text{Sum} = 5 \times (5+1) / 2 = 15$  .

##### **Example 2**

##### **Input:**

100

## Output:

5050

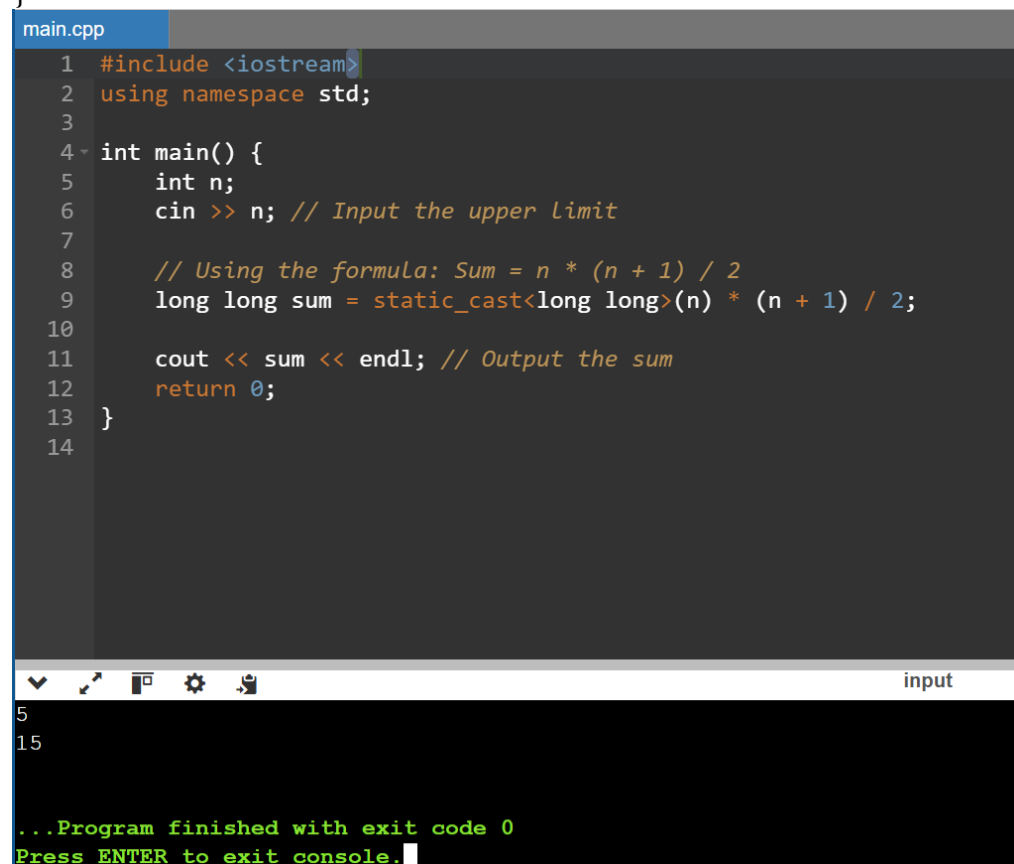
## Explanation:

Using the formula,  $\text{Sum} = 100 \times (100 + 1) / 2 = 5050$ .

## Code:

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

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



The screenshot shows a C++ IDE with a file named 'main.cpp'. The code is as follows:

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int n;
6     cin >> n; // Input the upper limit
7
8     // Using the formula: Sum = n * (n + 1) / 2
9     long long sum = static_cast<long long>(n) * (n + 1) / 2;
10
11     cout << sum << endl; // Output the sum
12     return 0;
13 }
14
```

The IDE's output window shows the input '5' and the resulting output '15'. Below the output, it states: '...Program finished with exit code 0' and 'Press ENTER to exit console.'

## 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  $\sqrt{n}$  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:

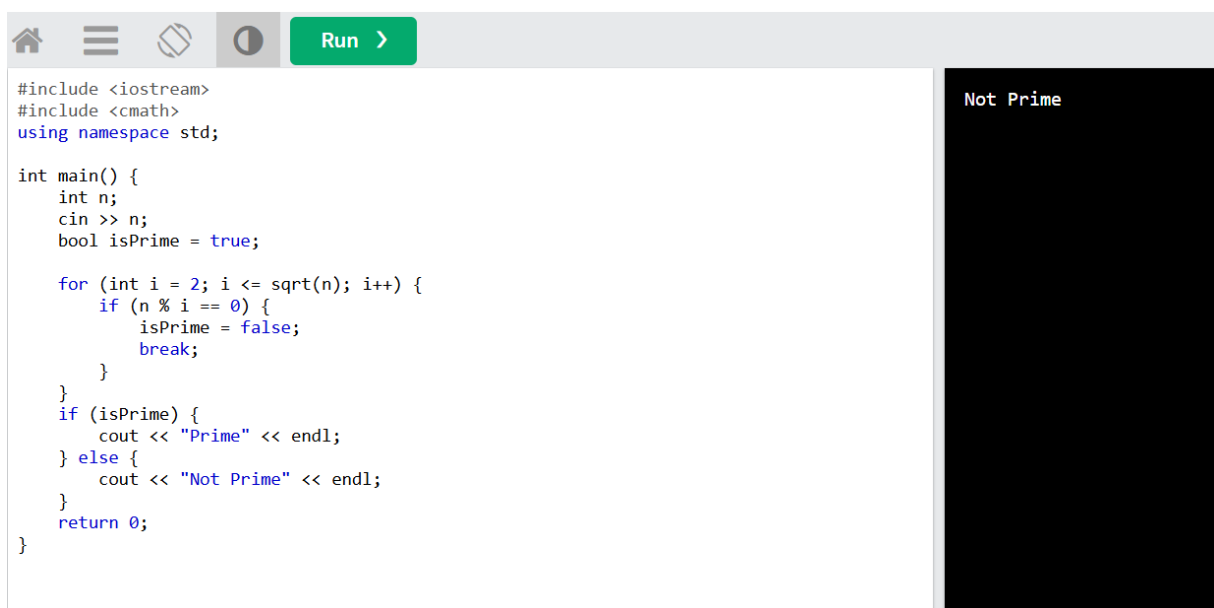
9 is divisible by 3, so it is not a prime number.

## Code:

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

int main() {
    int n;
    cin >> n;
    bool isPrime = true;
    for (int i = 2; i <= sqrt(n); i++) {
        if (n % i == 0) {
            isPrime = false;
            break;
        }
    }
    if (isPrime) {
        cout << "Prime" << endl;
    } else {
        cout << "Not Prime" << endl;
    }

    return 0;
}
```

A screenshot of a C++ IDE interface. The top bar contains icons for home, menu, undo, and redo, followed by a green 'Run' button with a right arrow. The main editor area displays the C++ code for checking if a number is prime. The code includes headers for iostream and cmath, uses the std namespace, and defines a main function that reads an integer n, checks for divisibility from 2 to sqrt(n), and prints 'Prime' or 'Not Prime'. The output window on the right shows 'Not Prime' on a black background.

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

int main() {
    int n;
    cin >> n;
    bool isPrime = true;

    for (int i = 2; i <= sqrt(n); i++) {
        if (n % i == 0) {
            isPrime = false;
            break;
        }
    }
    if (isPrime) {
        cout << "Prime" << endl;
    } else {
        cout << "Not Prime" << endl;
    }
    return 0;
}
```

Not Prime

### 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  $i \% 2 \neq 0$ .

#### 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

### Code:-

```
#include <iostream>
using namespace std;
int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; i += 2) {
        cout << i;
        if (i + 2 <= n) {
            cout << " ";
        }
    }
    cout << endl;
    return 0;
}
```

The screenshot displays an online C++ compiler interface. The code editor on the left contains the following code:

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int n;
6     cin >> n; // Input the upper limit
7
8     // Loop through numbers from 1 to n
9     for (int i = 1; i <= n; i += 2) {
10        cout << i;
11        if (i + 2 <= n) { // Add a space only if there's another number to print
12            cout << " ";
13        }
14    }
15
16    cout << endl; // End with a new line
17    return 0;
18 }
19
```

The right sidebar shows the execution results:

- Input:** 5
- Status:** Successfully executed
- Time:** 0.0000 secs
- Memory:** 3.528 Mb
- Sample Input:** 5
- Your Output:** 1 3 5

## 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.

##### **Example 2:**

##### **Input:**

900000

##### **Output:**

6

**Explanation:**

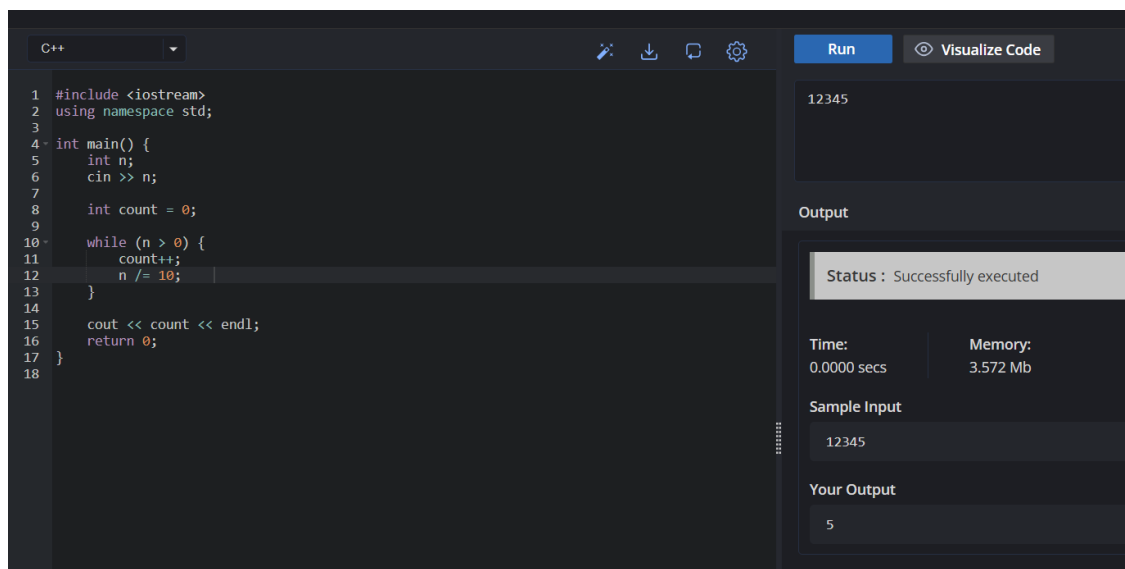
The number 900000 has 6 digits: 9, 0, 0, 0, 0, 0.

**Code:**

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

int main() {

    int n;
    cin >> n;
    int count = 0;
    while (n > 0) {
        count++;
        n /= 10;
    }
    cout << count << endl;
    return 0;
}
```



The screenshot shows a C++ IDE with the following code in the editor:

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int n;
6     cin >> n;
7
8     int count = 0;
9
10    while (n > 0) {
11        count++;
12        n /= 10;
13    }
14
15    cout << count << endl;
16    return 0;
17 }
18
```

The IDE interface includes a 'Run' button and a 'Visualize Code' button. The output panel on the right shows the following information:

- Input: 12345
- Status: Successfully executed
- Time: 0.0000 secs
- Memory: 3.572 Mb
- Sample Input: 12345
- Your Output: 5



## 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;

int main() {

    int n;
    cin >> n;

    int reversedNumber = 0;

    while (n > 0) {
        int lastDigit = n % 10;
        reversedNumber = reversedNumber * 10 + lastDigit;
        n /= 10;
    }

    cout << reversedNumber << endl;

    return 0;
}
```

## Output:

The screenshot displays an online C++ compiler interface. The code editor on the left contains the following C++ code:

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int n;
6     cin >> n;
7
8     int reversedNumber = 0;
9
10    while (n > 0) {
11        int lastDigit = n % 10;
12        reversedNumber = reversedNumber * 10 + lastDigit;
13        n /= 10;
14    }
15
16    cout << reversedNumber << endl;
17    return 0;
18 }
19
```

The right-hand side of the interface shows the execution results:

- Input:** 12345
- Output:** 54321
- Status:** Successfully executed
- Time:** 0.0000 secs
- Memory:** 3.388 Mb
- Sample Input:** 12345
- Your Output:** 54321

### 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.

#### Code:

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

int main() {

    int n;
    cin >> n;

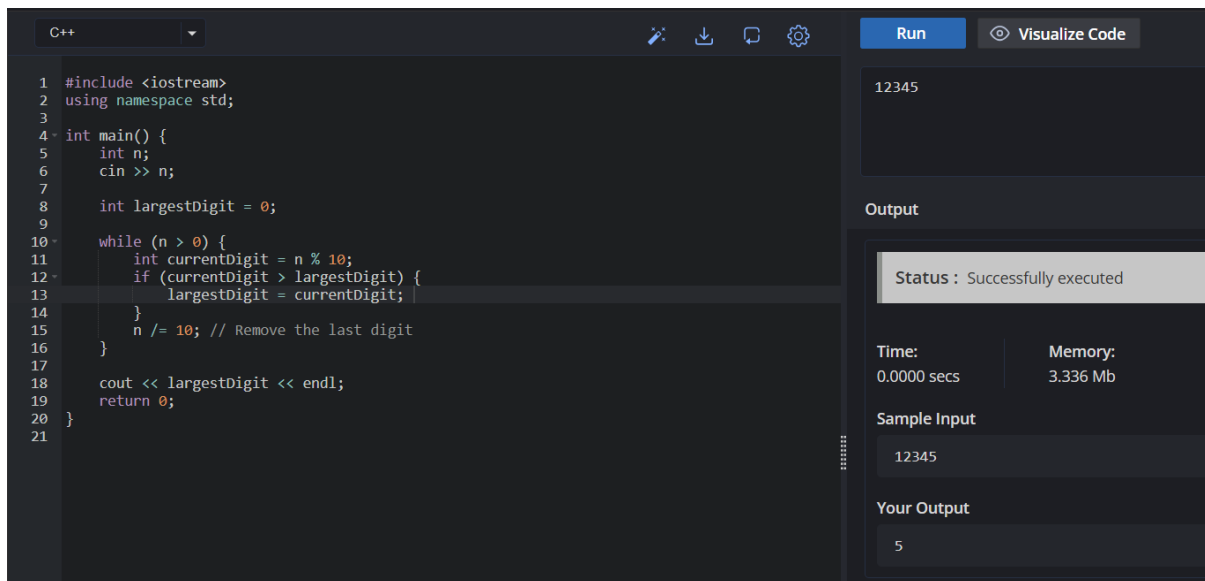
    int largestDigit = 0;

    while (n > 0) {
        int currentDigit = n % 10;
        if (currentDigit > largestDigit) {
            largestDigit = currentDigit;
        }
        n /= 10;
    }

    cout << largestDigit << endl;

    return 0;
}
```

### Output:



The screenshot shows a C++ IDE with the following code in the editor:

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int n;
6     cin >> n;
7
8     int largestDigit = 0;
9
10    while (n > 0) {
11        int currentDigit = n % 10;
12        if (currentDigit > largestDigit) {
13            largestDigit = currentDigit;
14        }
15        n /= 10; // Remove the last digit
16    }
17
18    cout << largestDigit << endl;
19    return 0;
20 }
21
```

The IDE interface includes a top bar with a language dropdown set to 'C++', icons for search, save, undo, redo, and settings, and buttons for 'Run' and 'Visualize Code'. The right sidebar displays the execution results:

- Output:** 12345
- Status:** Successfully executed
- Time:** 0.0000 secs
- Memory:** 3.336 Mb
- Sample Input:** 12345
- Your Output:** 5

## 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;
```

```
double calculateArea(double radius) {
    return 3.14159 * radius * radius;
```

```

}

int calculateArea(int length, int breadth) {
    return length * breadth;
}

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

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

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

    cout << calculateArea(radius) << endl;
    cout << calculateArea(length, breadth) << endl;
    cout << calculateArea(base, height) << endl;

    return 0;
}

```

### Output:-

The screenshot shows a C++ IDE with a code editor on the left and a console/output panel on the right. The code in the editor is a C++ program that calculates the area of a circle, a rectangle, and a triangle. The console shows the input '4 5 6 7 2' and the output '50.2654', '30', and '7'. The status bar indicates 'Status : Successfully executed'.

```

C++
1 #include <iostream>
2 using namespace std;
3
4 double calculateArea(double radius) {
5     return 3.14159 * radius * radius;
6 }
7
8 int calculateArea(int length, int breadth) {
9     return length * breadth;
10 }
11
12 double calculateArea(double base, double height) {
13     return 0.5 * base * height;
14 }
15
16 int main() {
17     double radius;
18     int length, breadth;
19     double base, height;
20
21     cin >> radius;
22     cin >> length >> breadth;
23     cin >> base >> height;
24
25     cout << calculateArea(radius) << endl;
26     cout << calculateArea(length, breadth) << endl;
27     cout << calculateArea(base, height) << endl;
28
29     return 0;
30 }
31

```

Run Visualize Code

4 5 6 7 2

Output

Status : Successfully executed

Time: 0.0000 secs Memory: 3.772 Mb

Sample Input

4 5 6 7 2

Your Output

50.2654  
30  
7

## 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

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

### 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:  $80000 + 30000 + 20000 =$

### Code:

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

int calculateSalary(int stipend) {
    return stipend;
}
```

```

int calculateSalary(int baseSalary, int bonuses) {
    return baseSalary + bonuses;
}

int calculateSalary(int baseSalary, int bonuses, int incentives) {
    return baseSalary + bonuses + incentives;
}

int main() {
    int stipend, baseSalary, bonuses, incentives;

    cin >> stipend;
    cin >> baseSalary >> bonuses;
    cout << "Intern Salary: " << calculateSalary(stipend) << endl;
    cout << "Employee Salary: " << calculateSalary(baseSalary, bonuses) << endl;

    cin >> baseSalary >> bonuses >> incentives;
    cout << "Manager Salary: " << calculateSalary(baseSalary, bonuses, incentives) << endl;

    return 0;
}

```

### Output:

The screenshot shows a C++ IDE with a code editor on the left and a console/output panel on the right. The code in the editor is a C++ program that calculates salaries based on base salary, bonuses, and incentives. The console shows the input values and the resulting salaries.

**Code Editor:**

```

1 #include <iostream>
2 using namespace std;
3
4 int calculateSalary(int stipend) {
5     return stipend;
6 }
7
8 int calculateSalary(int baseSalary, int bonuses) {
9     return baseSalary + bonuses;
10 }
11
12 int calculateSalary(int baseSalary, int bonuses, int incentives) {
13     return baseSalary + bonuses + incentives;
14 }
15
16 int main() {
17     int stipend, baseSalary, bonuses, incentives;
18
19     cin >> stipend;
20     cin >> baseSalary >> bonuses;
21     cout << "Intern Salary: " << calculateSalary(stipend) << endl;
22     cout << "Employee Salary: " << calculateSalary(baseSalary, bonuses) << endl;
23
24     cin >> baseSalary >> bonuses >> incentives;
25     cout << "Manager Salary: " << calculateSalary(baseSalary, bonuses, incentives) << endl;
26
27     return 0;
28 }
29

```

**Console/Output Panel:**

10000 50000 20000 80000 30000 20000

**Output**

Status : Successfully executed

Time: 0.0000 secs Memory: 3.528 Mb

**Sample Input**

10000 50000 20000 80000 30000 20000

**Your Output**

Intern Salary: 10000  
Employee Salary: 70000  
Manager Salary: 130000



### 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*

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

#### *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

**Code:**

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

class Employee {
private:
    int id;
    string name;
    float salary;

public:
    void setID(int empID) {
        id = empID;
    }

    void setName(string empName) {
        name = empName;
    }

    void setSalary(float empSalary) {
        salary = empSalary;
    }

    int getID() {
        return id;
    }

    string getName() {
        return name;
    }

    float getSalary() {
        return salary;
    }

    void displayDetails() {
        cout << "Employee ID: " << id << endl;
        cout << "Employee Name: " << name << endl;
        cout << "Employee Salary: " << salary << endl;
    }
};

int main() {
    Employee emp;
    int id;
```

```

    string name;
    float salary;

    cin >> id;
    cin.ignore();
    getline(cin, name);
    cin >> salary;

    emp.setID(id);
    emp.setName(name);
    emp.setSalary(salary);

    emp.displayDetails();

    return 0;
}

```

## Output:

The screenshot shows a C++ IDE with a dark theme. The left pane displays the source code for an `Employee` class and its `main` function. The right pane shows the program's execution output, including sample input and the resulting output.

```

C++
16 void setSalary(float empSalary) {
17     name = empName;
18 }
19
20 void setSalary(float empSalary) {
21     salary = empSalary;
22 }
23
24 int getID() {
25     return id;
26 }
27
28 string getName() {
29     return name;
30 }
31
32 float getSalary() {
33     return salary;
34 }
35
36 void displayDetails() {
37     cout << "Employee ID: " << id << endl;
38     cout << "Employee Name: " << name << endl;
39     cout << "Employee Salary: " << salary << endl;
40 }
41 };
42
43 int main() {
44     Employee emp;
45     int id;
46     string name;
47     float salary;
48
49     cin >> id;
50     cin.ignore();
51     getline(cin, name);
52     cin >> salary;
53

```

**Run** **Visualize Code**

```

202 Jane Smith
85000.75

```

**Output**

Status : Successfully executed

Time:	Memory:
0.0000 secs	3.748 Mb

**Sample Input**

```

202 Jane Smith
85000.75

```

**Your Output**

```

Employee ID: 202
Employee Name: Jane Smith
Employee Salary: 85000.8

```

## **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*

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

#### *Output Format*

Print the computed area of each shape 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  $3.14159 \times 5^2 = 78.53975$ .
- The area of the rectangle is  $4 \times 6 = 24$ .
- The area of the triangle is  $0.5 \times 3 \times 7 = 10.5$ .

Code:

```
#include <iostream>
```

```
#include <string>
```

```
using namespace std;
```

```
class Employee {
```

```
private:
```

```
    int id;
```

```

    string name;
    float salary;

public:
    void setID(int empID) {
        id = empID;
    }

    void setName(string empName) {
        name = empName;
    }

    void setSalary(float empSalary) {
        salary = empSalary;
    }

    int getID() {
        return id;
    }

    string getName() {
        return name;
    }

    float getSalary() {
        return salary;
    }

    void displayDetails() {
        cout << "Employee ID: " << id << endl;
        cout << "Employee Name: " << name << endl;
        cout << "Employee Salary: " << salary << endl;
    }
};

int main() {
    Employee emp;
    int id;
    string name;
    float salary;

    cin >> id;
    cin.ignore();
    getline(cin, name);
    cin >> salary;

    emp.setID(id);
    emp.setName(name);
    emp.setSalary(salary);

```

```

emp.displayDetails();

return 0;
}

```

## Output:

The screenshot shows a C++ IDE with the following code in the editor:

```

1  #include <iostream>
2  using namespace std;
3
4  class Shape {
5  public:
6      virtual double area() = 0; // Pure virtual function
7  };
8
9  class Circle : public Shape {
10 private:
11     double radius;
12 public:
13     Circle(double r) : radius(r) {}
14     double area() override {
15         return 3.14159 * radius * radius;
16     }
17 };
18
19 class Rectangle : public Shape {
20 private:
21     double length, breadth;
22 public:
23     Rectangle(double l, double b) : length(l), breadth(b) {}
24     double area() override {
25         return length * breadth;
26     }
27 };
28
29 class Triangle : public Shape {
30 private:
31     double base, height;
32 public:
33     Triangle(double b, double h) : base(b), height(h) {}
34     double area() override {
35         return 0.5 * base * height;
36     }
37 };
38

```

The right sidebar shows the execution output:

**Run** **Visualize Code**

5  
4 6  
3 7

**Output**

Status : Successfully executed

Time: 0.0000 secs      Memory: 3.728 Mb

**Sample Input**

5  
4 6  
3 7

**Your Output**

Area of Circle: 78.5397  
Area of Rectangle: 24  
Area of Triangle: 10.5

## 2) 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  $1 \leq \text{choice} \leq 3$  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.

### **Test Cases:**

#### *Example 1: Circle*

##### **Input:**

Choice: 1

Radius: 7.5

##### **Output:**

Shape: Circle

Radius: 7.5

Area: 176.714

##### **Code:**

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

class Shape {
public:
    void calculateArea(int choice) {
        if (choice == 1) {
            float radius;
            cout << "Enter radius of the circle: ";
            cin >> radius;
            if (radius > 0) {
                cout << "Shape: Circle" << endl;
                cout << "Radius: " << radius << endl;
                cout << "Area: " << calculateArea(radius) << endl;
            } else {
                cout << "Invalid input for radius. Must be greater than 0." << endl;
            }
        } else if (choice == 2) {
            float length, breadth;
            cout << "Enter length and breadth of the rectangle: ";
            cin >> length >> breadth;
            if (length > 0 && breadth > 0) {
                cout << "Shape: Rectangle" << endl;
                cout << "Length: " << length << ", Breadth: " << breadth << endl;
                cout << "Area: " << calculateArea(length, breadth) << endl;
            } else {
                cout << "Invalid input for length or breadth. Must be greater than 0." << endl;
            }
        } else if (choice == 3) {
            float base, height;
            cout << "Enter base and height of the triangle: ";
            cin >> base >> height;
            if (base > 0 && height > 0) {
                cout << "Shape: Triangle" << endl;
                cout << "Base: " << base << ", Height: " << height << endl;
                cout << "Area: " << calculateArea(base, height) << endl;
            } else {
                cout << "Invalid input for base or height. Must be greater than 0." << endl;
            }
        }
    }
};
```

```

        cout << "Invalid input for base or height. Must be greater than 0." << endl;
    }
} else {
    cout << "Invalid choice." << endl;
}
}

private:
    float calculateArea(float radius) {
        return 3.14159 * radius * radius;
    }

    float calculateArea(float length, float breadth) {
        return length * breadth;
    }

    float calculateArea(float base, float height) {
        return 0.5 * base * height;
    }
};

int main() {
    int choice;
    cout << "Enter your choice (1 for Circle, 2 for Rectangle, 3 for Triangle): ";
    cin >> choice;

    Shape shape;
    shape.calculateArea(choice);

    return 0;
}

```

### Output:

mathematica

```

Enter your choice (1 for Circle, 2 for Rectangle, 3 for Triangle): 2
Enter length and breadth of the rectangle: 4 6

```

### Output:

yaml

```

Shape: Rectangle
Length: 4, Breadth: 6
Area: 24

```



#### 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:  $1 \leq \text{type} \leq 3$ .
- Length, Width, Radius, Base, and Height:  $1.0 \leq \text{value} \leq 10^4$ .
- Use  $\pi=3.14159$  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

###### **Code:**

```
#include <iostream>
```

```
#include <cmath>
```

```

#include <iomanip> // For controlling output precision
using namespace std;

class Shape {
public:
    virtual void calculateArea() = 0; // Pure virtual function for calculating area
    virtual ~Shape() {} // Virtual destructor to ensure proper cleanup
};

class Rectangle : public Shape {
private:
    float length, width;
public:
    void setDimensions(float l, float w) {
        length = l;
        width = w;
    }

    void calculateArea() override {
        cout << "Shape: Rectangle" << endl;
        cout << "Area: " << fixed << setprecision(2) << length * width << endl;
    }
};

class Circle : public Shape {
private:
    float radius;
public:
    void setRadius(float r) {
        radius = r;
    }

    void calculateArea() override {
        cout << "Shape: Circle" << endl;
        cout << "Area: " << fixed << setprecision(2) << 3.14159 * radius * radius << endl;
    }
};

class Triangle : public Shape {
private:
    float base, height;
public:
    void setDimensions(float b, float h) {
        base = b;
        height = h;
    }

    void calculateArea() override {
        cout << "Shape: Triangle" << endl;
        cout << "Area: " << fixed << setprecision(2) << 0.5 * base * height << endl;
    }
};

int main() {
    int shapeType;

```

```

Shape* shape = nullptr;

cout << "Enter shape type (1 for Rectangle, 2 for Circle, 3 for Triangle): ";
cin >> shapeType;

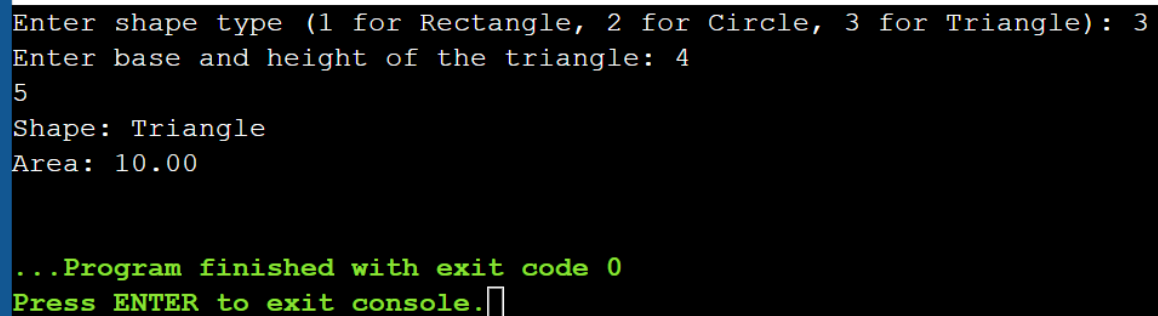
if (shapeType == 1) {
    float length, width;
    cout << "Enter length and width of the rectangle: ";
    cin >> length >> width;
    shape = new Rectangle();
    dynamic_cast<Rectangle*>(shape)->setDimensions(length, width);
}
else if (shapeType == 2) {
    float radius;
    cout << "Enter radius of the circle: ";
    cin >> radius;
    shape = new Circle();
    dynamic_cast<Circle*>(shape)->setRadius(radius);
}
else if (shapeType == 3) {
    float base, height;
    cout << "Enter base and height of the triangle: ";
    cin >> base >> height;
    shape = new Triangle();
    dynamic_cast<Triangle*>(shape)->setDimensions(base, height);
}
else {
    cout << "Invalid shape type!" << endl;
    return 1;
}

shape->calculateArea(); // Polymorphic call

delete shape; // Free the allocated memory
return 0;
}

```

### Output:



```

Enter shape type (1 for Rectangle, 2 for Circle, 3 for Triangle): 3
Enter base and height of the triangle: 4
5
Shape: Triangle
Area: 10.00

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

```