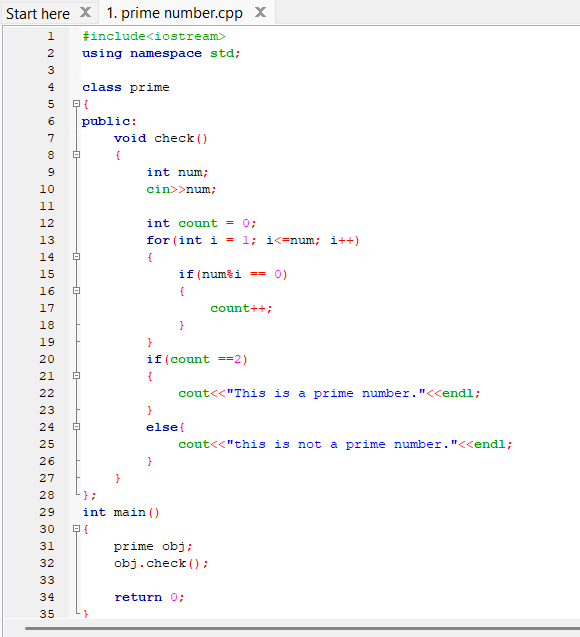
**LAB-1**

**Title of the problem: PRIME NUMBER**

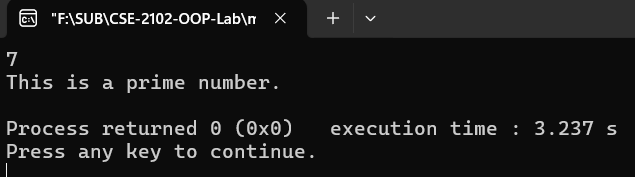
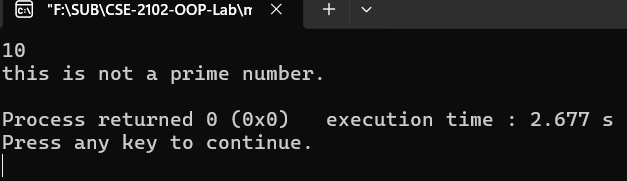
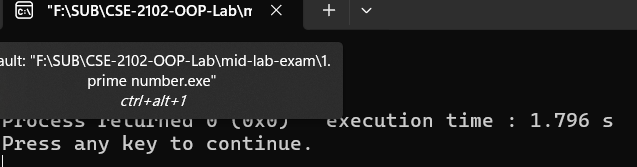
**Test case analysis:**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 7 | This is a prime number. |
| 10 | This is a prime number. |
| 1 | This is not a prime number. |

**Coding:**

****

**Result:**

**** **** 

**Justification of the output:**

1. **Input: 7 → Output: "This is a prime number."**
   * The number 7 is only divisible by **1 and 7**.
   * The total count of divisors is **2**, which satisfies the definition of a **prime number**.
   * So, the code correctly prints: **"This is a prime number."**
2. **Input: 10 → Output: "this is not a prime number."**
   * The number 10 is divisible by **1, 2, 5, and 10**.
   * The total count of divisors is **4**, which is more than 2.
   * Therefore, 10 is **not a prime number**, and the code correctly identifies that.
3. **Input: 1 → Output: "this is not a prime number."**
   * The number 1 is divisible by **only 1**.
   * Prime numbers must have exactly **2 distinct positive divisors**.
   * Since 1 has only **1 divisor**, it is **not prime** by definition.
   * So, the output is correct.

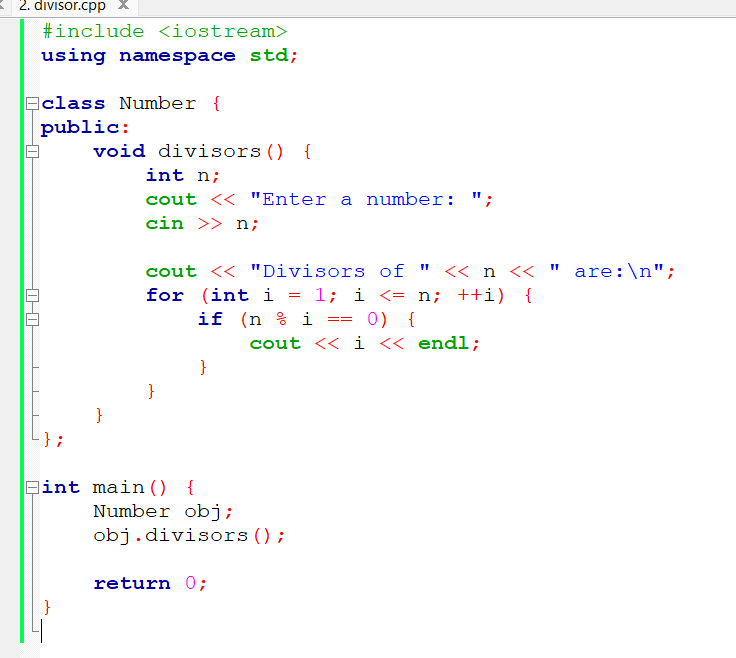
**LAB-2**

**Title of the problem: DIVISOR**

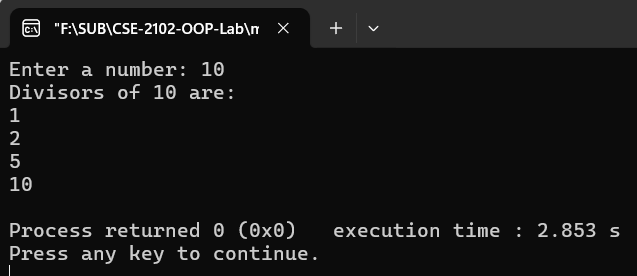
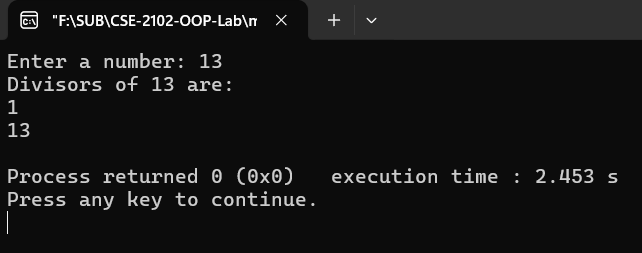
**Test case analysis:**

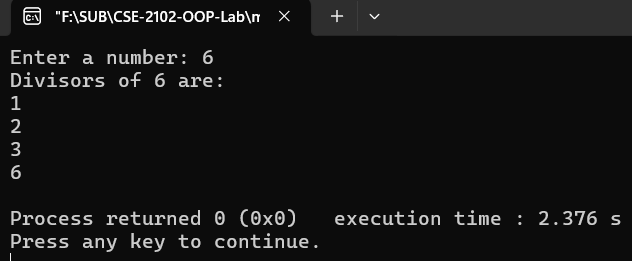
|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 6 | 1,2,5,10 |
| 10 | 1,2,3,6 |
| 13 | 1,13 |

**Coding:**

****

**Result:**

**** ****

****

**Justification of the output:**

 **Input: 6 → Output: 1, 2, 3, 6**

* The divisors of 6 are: **1, 2, 3, and 6**, because 6 is divisible by all of them.
* The code checks every number from 1 to n using if(n % i == 0) and prints those that divide n with no remainder.

 **Input: 10 → Output: 1, 2, 5, 10**

* 10 is divisible by **1, 2, 5, and 10**.
* So, all these values are printed as valid divisors.

 **Input: 13 → Output: 1, 13**

* 13 is a **prime number**, so it only has two divisors: **1 and itself (13)**.
* Hence, only those two values are printed.

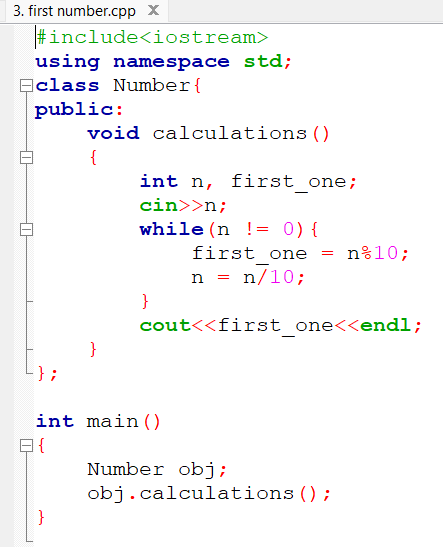
**LAB-3**

**Title of the problem: FIRST NUMBERT**

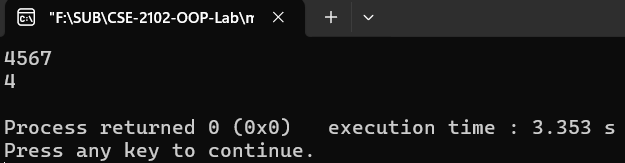
**Test case analysis:**

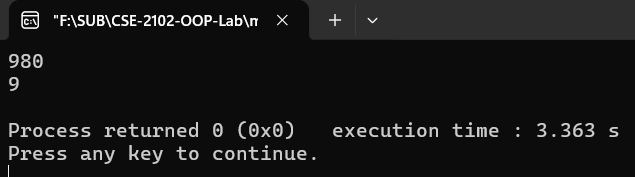
|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 4567 | 4 |
| 980 | 9 |
| 5 | 5 |

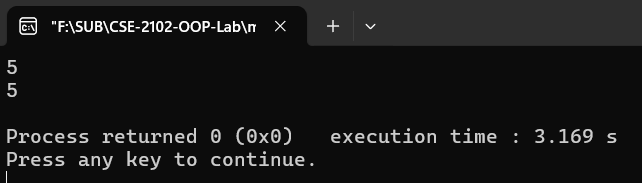
**Coding:**

****

**Result:**

****

****

****

**Justification of the output:**

 **Input: 4567 → Output: 4**

* The number is processed digit by digit from the **last digit to the first** using n % 10.
* first\_one is updated in each iteration until n becomes 0.
* At the end of the loop, first\_one holds the **first digit**, which is **4** in this case.

 **Input: 980 → Output: 9**

* Digits from last to first: 0, 8, 9.
* Final value stored in first\_one is **9**, the first digit.

 **Input: 5 → Output: 5**

* Only one digit, so the loop runs once.
* first\_one = 5, and that’s the correct output.

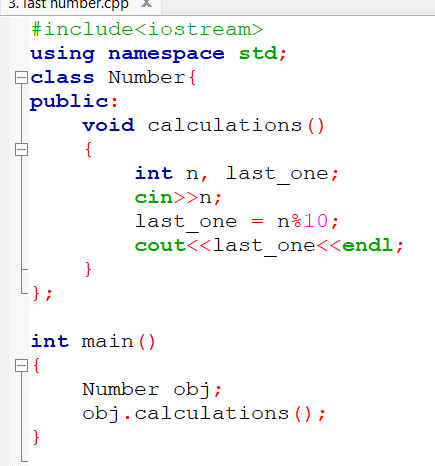
**LAB-3**

**Title of the problem: LAST NUMBER**

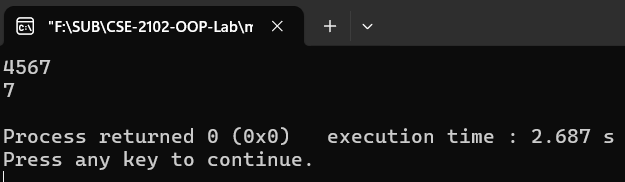
**Test case analysis:**

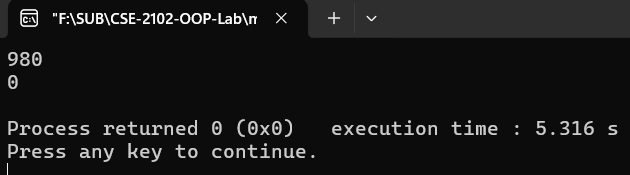
|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 4767 | 7 |
| 980 | 0 |
| 5 | 5 |

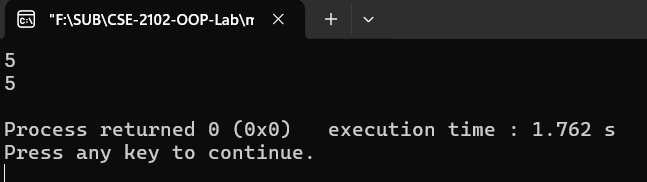
**Coding:**

****

**Result:**

****

****

****

**Justification of the output:**

 **Input: 4567 → Output: 7**

* The last digit of 4567 is found using 4567 % 10 = 7.
* So, it correctly prints **7**.

 **Input: 980 → Output: 0**

* 980 % 10 = 0 → last digit is **0**.

 **Input: 5 → Output: 5**

* A single-digit number returns itself: 5 % 10 = 5.

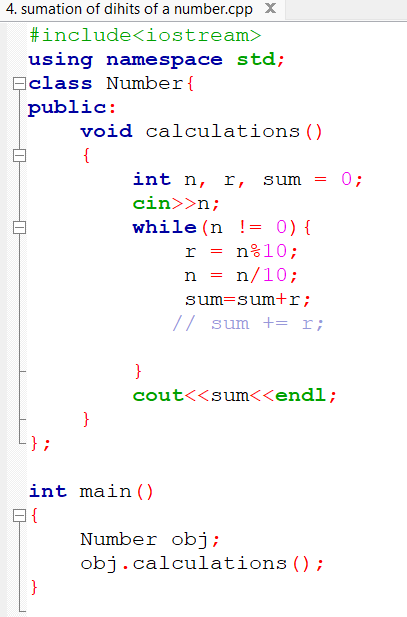
**LAB-4**

**Title of the problem: SUMATION OF DIGITS**

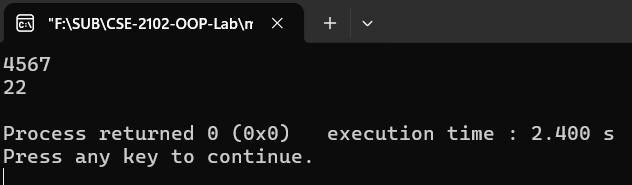
**Test case analysis:**

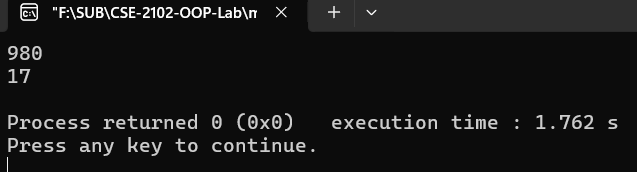
|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 4567 | 22 |
| 980 | 17 |
| 5 | 5 |

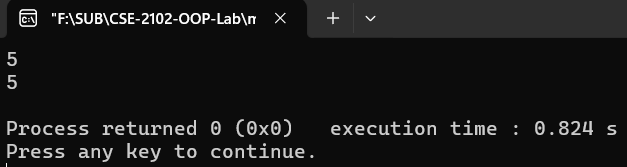
**Coding:**

****

**Result:**

****

****

****

**Justification of the output:**

 **Input: 4567 → Output: 22**

* Digits: 4 + 5 + 6 + 7 = **22**
* Extracted using r = n % 10 in a loop, and added to sum.

 **Input: 980 → Output: 17**

* Digits: 9 + 8 + 0 = **17**

 **Input: 5 → Output: 5**

* Single digit: 5 → sum is **5**

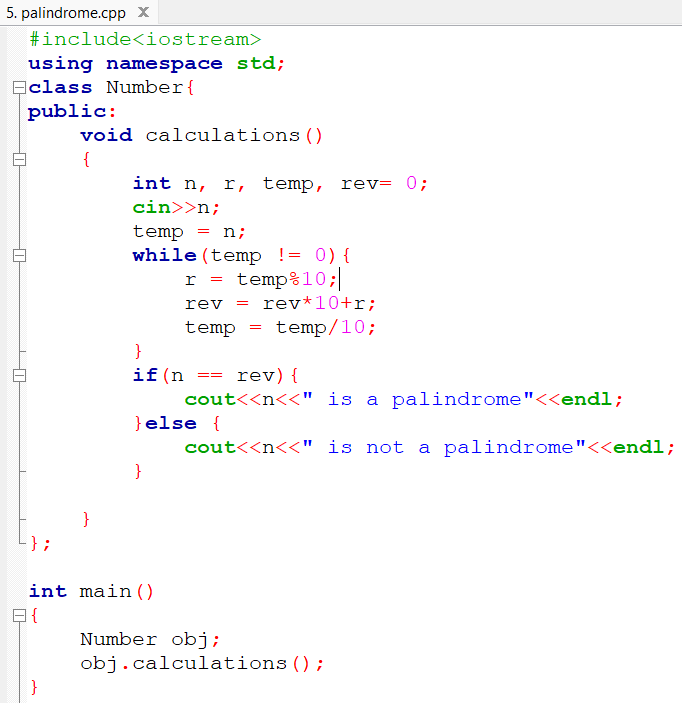
**LAB-5**

**Title of the problem: PALINDROME**

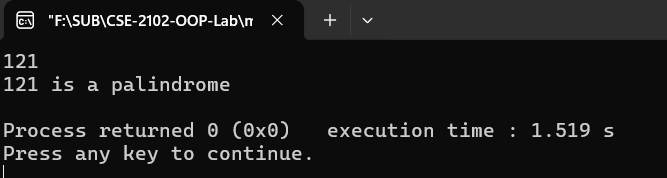
**Test case analysis:**

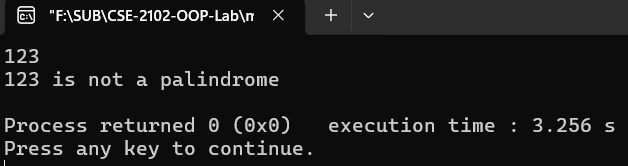
|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 121 | **121 is a palindrome** |
| 123 | **123 is not a palindrome** |
| 7 | **7 is a palindrome** |

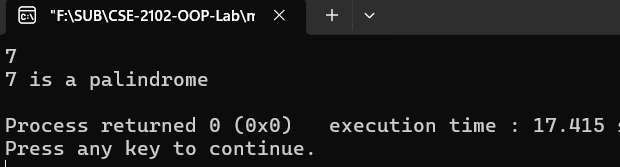
**Coding:**

****

**Result:**

****

****

****

**Justification of the output:**

 **Input: 121 → Output: 121 is a palindrome**

* Reverse of 121 = 121 → original and reversed values match.
* So, the condition n == rev is true.

 **Input: 123 → Output: 123 is not a palindrome**

* Reverse of 123 = 321 → does not match original.
* Hence, the output is correct.

 **Input: 7 → Output: 7 is a palindrome**

* Single-digit numbers are **always palindromes** (they read the same both ways).
* So n == rev holds true.

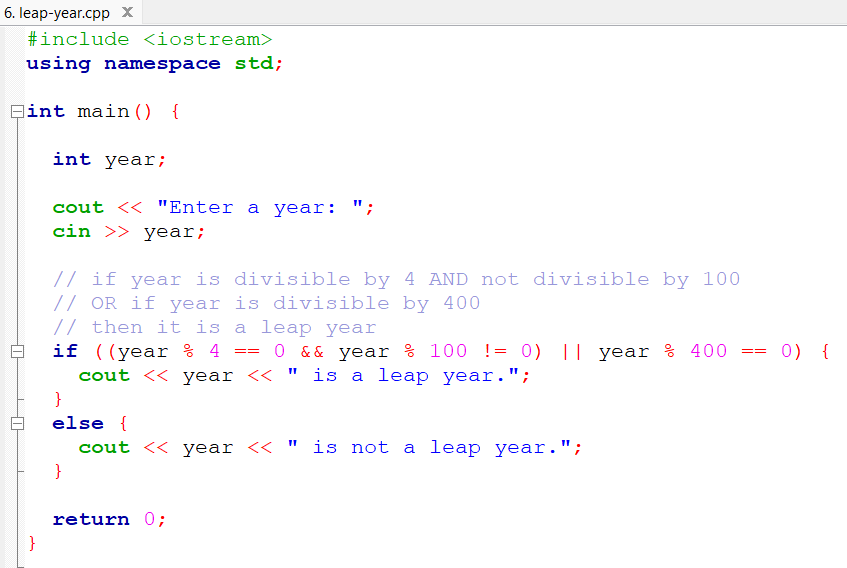
**LAB-6**

**Title of the problem:**

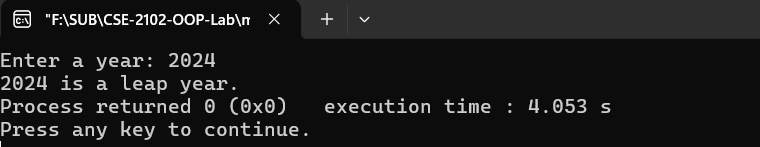
**Test case analysis:**

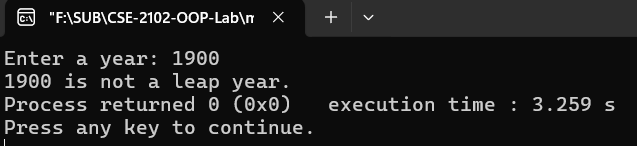
|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 2024 | **2024 is a leap year.** |
| 1900 | **1900 is not a leap year.** |
| 2000 | **2000 is a leap year.** |

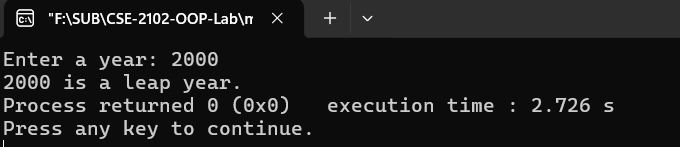
**Coding:**

****

**Result:**

****

****

****

**Justification of the output:**

 **Input: 2024 → Output: 2024 is a leap year.**

* 2024 is divisible by 4 → ✅
* Not divisible by 100 → ✅
* So it meets leap year condition.

 **Input: 1900 → Output: 1900 is not a leap year.**

* 1900 is divisible by 4 → ✅
* Divisible by 100 → ✅
* Not divisible by 400 → ❌
* So it's **not** a leap year.

 **Input: 2000 → Output: 2000 is a leap year.**

* Divisible by 4 → ✅
* Divisible by 100 → ✅
* Divisible by 400 → ✅
* So it's a valid leap year.

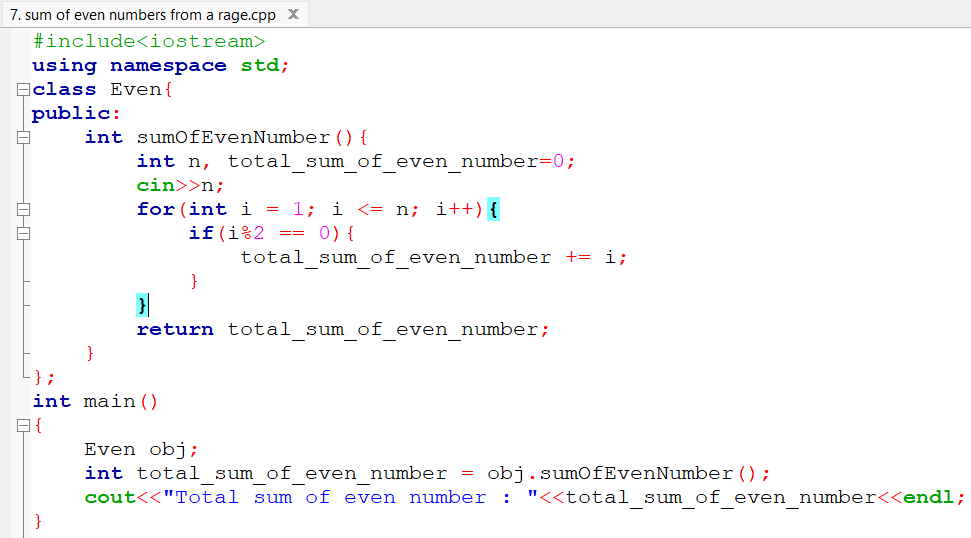
**LAB-7**

**Title of the problem: SUM OF EVEN NUMBER FROM A RANGE**

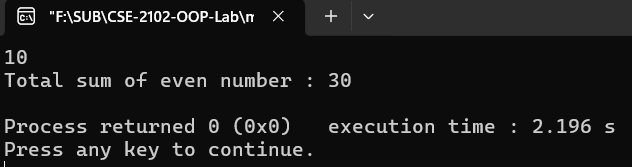
**Test case analysis:**

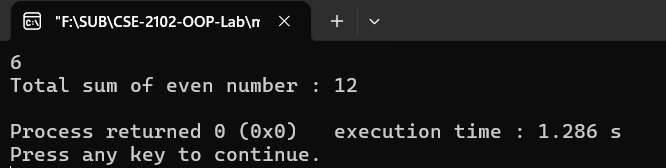
|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 10 | 30 |
| 6 | 12 |
| 1 | 0 |

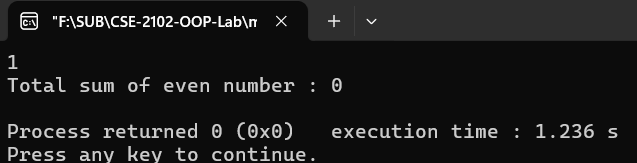
**Coding:**

****

**Result:**

****

****

****

**Justification of the output:**

 **Input: 10 → Output: 30**

* Even numbers between 1 and 10: **2 + 4 + 6 + 8 + 10 = 30**
* The loop checks each number i from 1 to n, and if i % 2 == 0, it adds i to the total.

 **Input: 6 → Output: 12**

* Even numbers: **2 + 4 + 6 = 12**

 **Input: 1 → Output: 0**

* No even number between 1 and 1 → sum = **0**

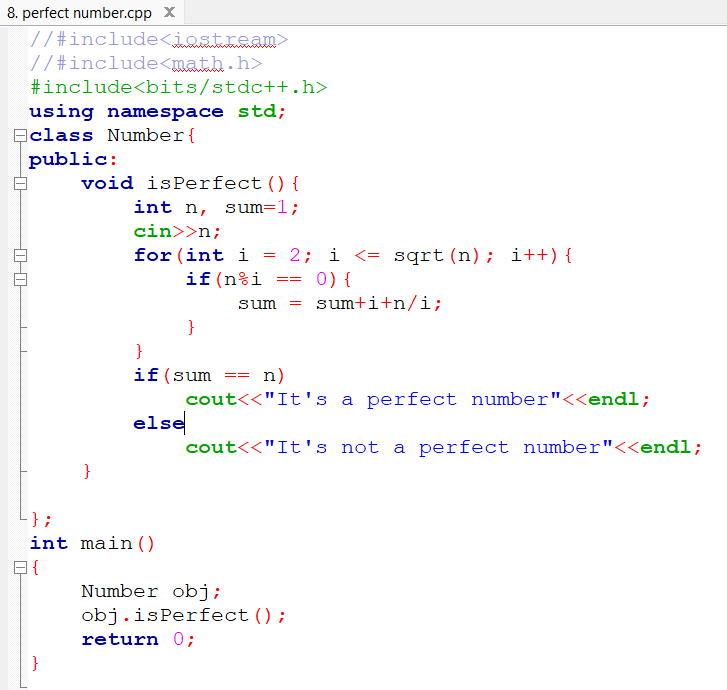
**LAB-8**

**Title of the problem: PERFECT NUMBER**

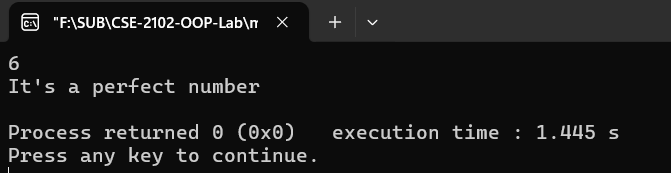
**Test case analysis:**

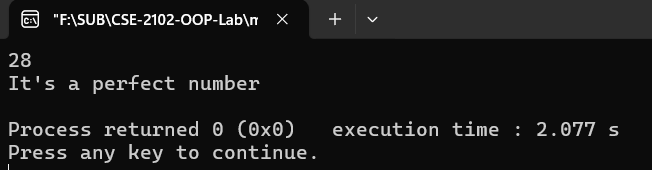
|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 6 | **It's a perfect number** |
| 28 | **It's a perfect number** |
| 12 | **It's not a perfect number** |

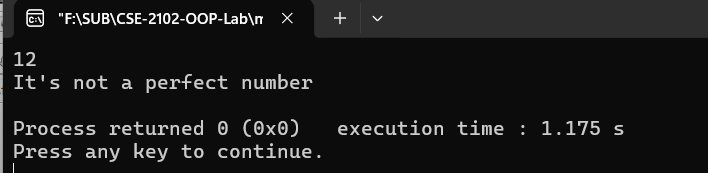
**Coding:**

****

**Result:**

****

****

****

**Justification of the output:**

 **Input: 6 → Output: It's a perfect number**

* Proper divisors of 6: 1, 2, 3
* Sum = 1 + 2 + 3 = **6**
* So it satisfies the condition for a perfect number.

 **Input: 28 → Output: It's a perfect number**

* Proper divisors: 1, 2, 4, 7, 14
* Sum = 1 + 2 + 4 + 7 + 14 = **28**

 **Input: 12 → Output: It's not a perfect number**

* Proper divisors: 1, 2, 3, 4, 6
* Sum = 1 + 2 + 3 + 4 + 6 = **16** ≠ 12

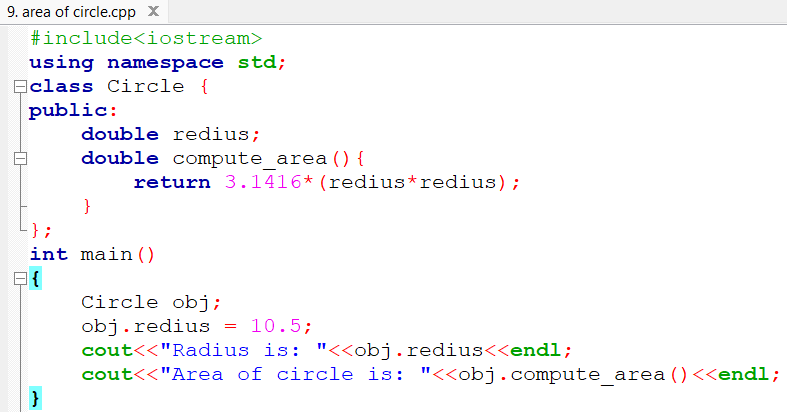
**LAB-9**

**Title of the problem: AREA OF CIRCLE**

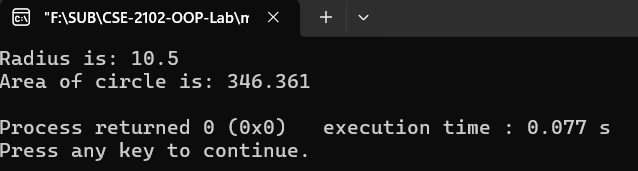
**Test case analysis:**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 10.5 | **Radius is: 10.5, Area of circle is: 346.36** |

**Coding:**

****

**Result:**

****

**Justification of the output:**

**Input: 10.5 → Output: Radius is: 10.5, Area of circle is: 346.36**

* The radius is assigned to the redius attribute of the Circle class.
* The area of the circle is calculated using the formula:

Area=π×r2=3.1416×(10.5×10.5)=346.36\text{Area} = \pi \times r^2 = 3.1416 \times (10.5 \times 10.5) = 346.36Area=π×r2=3.1416×(10.5×10.5)=346.36

* The code correctly calculates the area and prints the results.

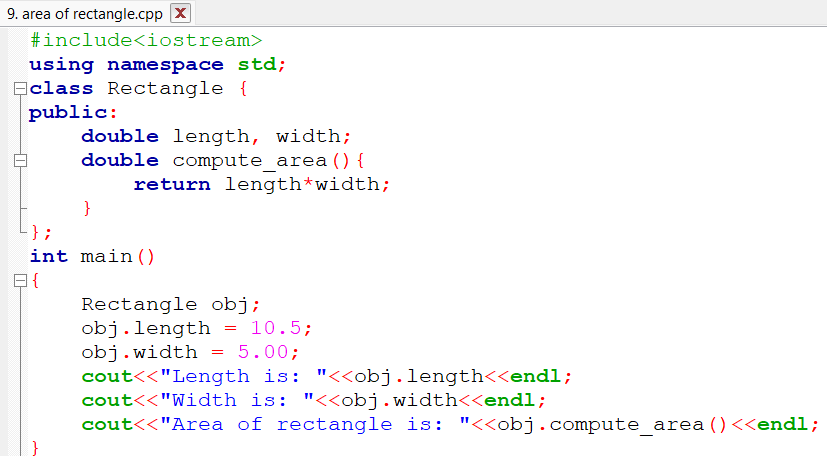
**LAB-9**

**Title of the problem: AREA OF RECTANGLE**

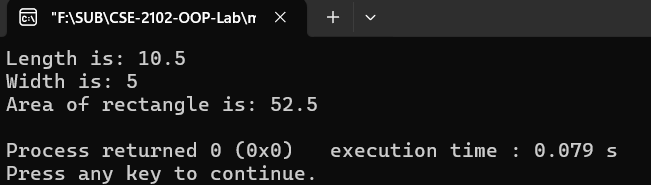
**Test case analysis:**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 10.5,5.00 | **Width is: 5.00, Area of rectangle is: 52.5** |

**Coding:**

****

**Result:**

****

**Justification of the output:**

**Input: Length = 10.5, Width = 5.00 → Output: Length is: 10.5, Width is: 5.00, Area of rectangle is: 52.5**

* The **length** of the rectangle is set to **10.5** and the **width** is set to **5.0**.
* The area of the rectangle is calculated using the formula:

Area=Length×Width=10.5×5=52.5\text{Area} = \text{Length} \times \text{Width} = 10.5 \times 5 = 52.5Area=Length×Width=10.5×5=52.5

* The code outputs the correct length, width, and area based on this formula.

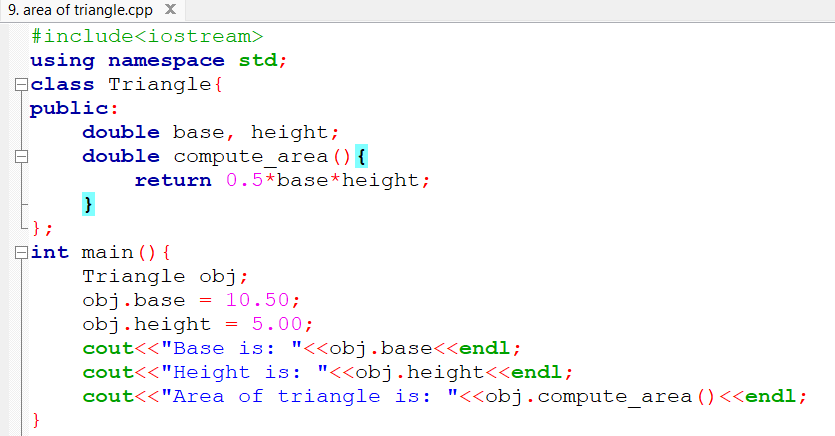
**LAB-9**

**Title of the problem: AREA OF RECTANGLE**

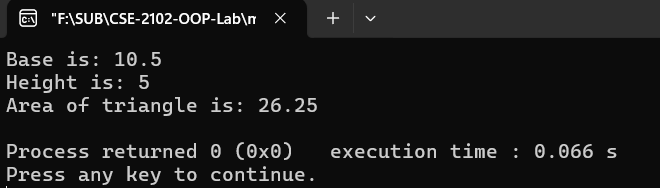
**Test case analysis:**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 10.5,5 | 26.25 |

**Coding:**

****

**Result:**

****

**Justification of the output:**

**Input: Base = 10.5, Height = 5 → Output: Area = 26.25**

* Area of a triangle is calculated using the formula:

Area=12×base×height\text{Area} = \frac{1}{2} \times \text{base} \times \text{height}Area=21​×base×height =0.5×10.5×5=26.25= 0.5 \times 10.5 \times 5 = 26.25=0.5×10.5×5=26.25

* The code sets the values, calls compute\_area(), and correctly prints the result

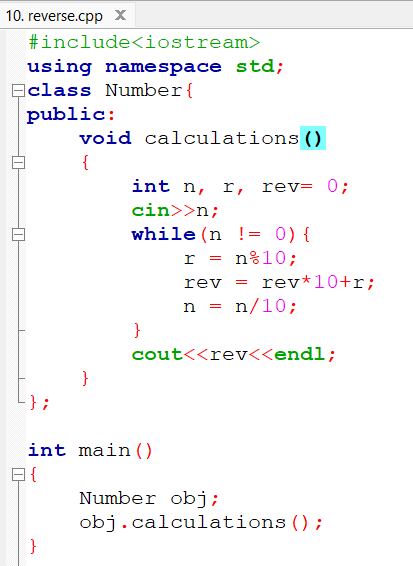
**LAB-10**

**Title of the problem: REVERSE**

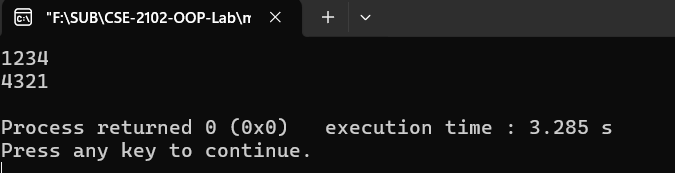
**Test case analysis:**

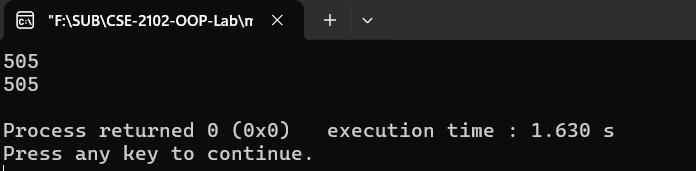
|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 1234 | 4321 |
| 505 | 505 |
| 900 | 9 |

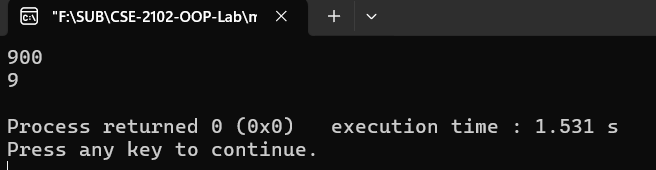
**Coding:**

****

**Result:**

****

****

****

**Justification of the output:**

 **Input: 1234 → Output: 4321**

* The digits are reversed step by step:  
  1234 → 4 → 43 → 432 → 4321

 **Input: 505 → Output: 505**

* Reversed digits: 5 → 50 → 505
* Leading zero in middle doesn't affect the reversed number.

 **Input: 900 → Output: 9**

* Reversed steps: 0 → 0 → 9
* Leading zeros (00) are dropped in the integer output, so result is just **9**

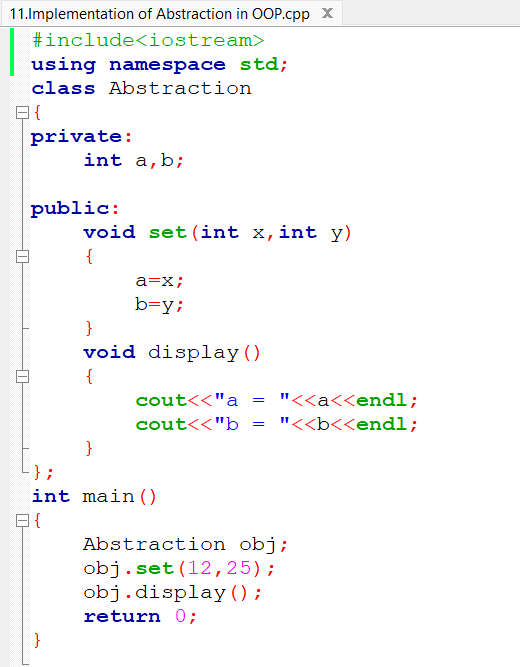
**LAB-11**

**Title of the problem: INPLEMENTATION OF ABSTRACTION IN OOP.**

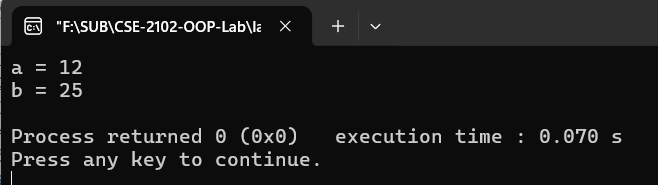
**Test case analysis:**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| Set(12,25) | a=12  b=25 |

**Coding:**

****

**Result:**

****

**Justification of the output:**

 **Abstraction** is implemented here by **hiding internal data (a, b)** and **exposing only necessary functions (set() and display())**.

 The data members a and b are marked as private, which means:

* They cannot be accessed or modified directly from outside the class.
* Only set() can assign values, and display() can show them.

 When obj.set(12, 25) is called:

* It sets a = 12, b = 25.

 Then obj.display() prints:

a = 12

b = 25

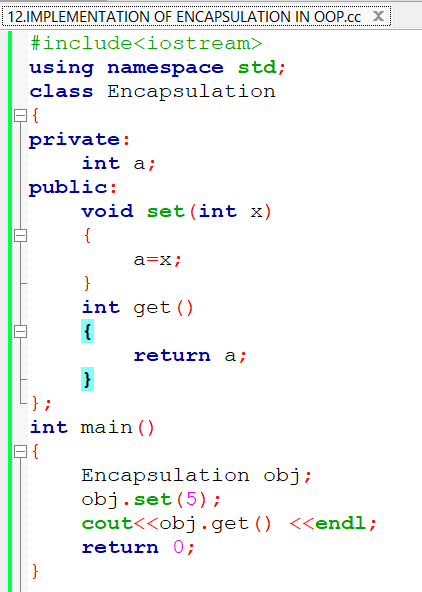
**LAB-12**

**Title of the problem: INPLEMENTATION OF ENCAPSULATION IN OOP.**

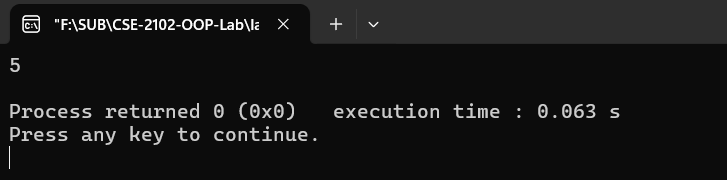
**Test case analysis:**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| Set(5) | 5 |

**Coding:**

****

**Result:**

****

**Justification of the output:**

1. **Encapsulation** means **binding data and methods** that operate on that data within one class, and **restricting direct access** to some components.
2. Here, the variable a is **private**, so it cannot be accessed directly from outside the class.
3. The method set(int x) assigns the value 5 to a.
4. The method get() returns the value of a, which is then printed

5

**LAB-13**

**Title of the problem: IMPLEMENTATION OF SOLUTION OF DIAMOND PROBLEM IN OOP.**

**Test case analysis:**

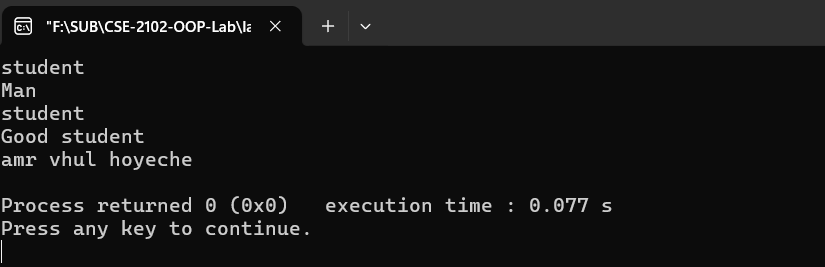
|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| Monir h;  (without virtual inheritance) | student Man student Good student amr vhul hoyeche |
| Monir h;  (with virtual inheritance) | student Man Good student amr vhul hoyeche |

**Coding:**

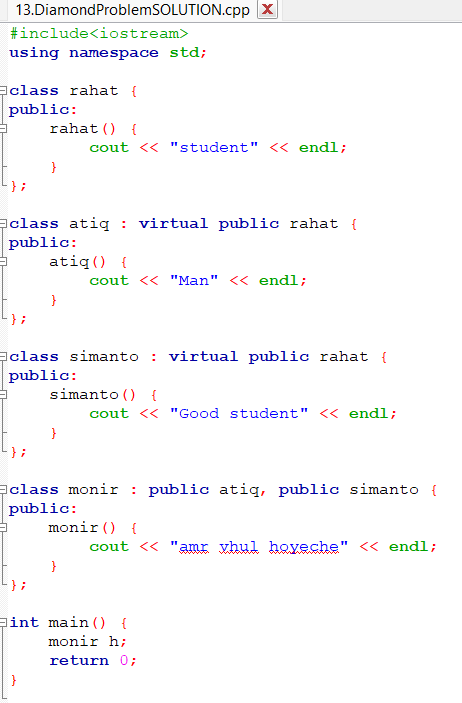
**i.Implementation**

****

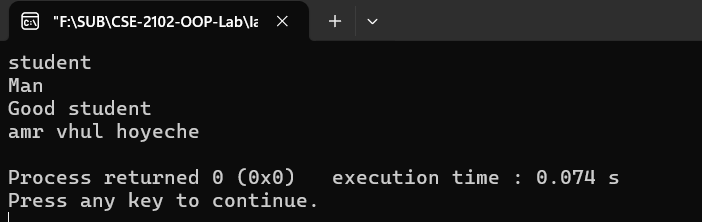
**Implementation result:**

****

**ii.Solution:**

****

**Solution Result:**

****

**Justification of the output:**

**🔴 Without virtual inheritance (Implementation Version)**

class atiq : public rahat

class simanto : public rahat

* **monir** inherits from both atiq and simanto.
* Both atiq and simanto **individually inherit** from rahat.
* So when monir is constructed:
  + rahat() is called via atiq.
  + rahat() is called again via simanto.
  + That’s why **student** is printed **twice**.
* Then constructors execute in order:
  + Man
  + Good student
  + amr vhul hoyeche

**✅ With virtual inheritance (Solution Version)**

class atiq : virtual public rahat

class simanto : virtual public rahat

* Now, both atiq and simanto **share a single instance** of rahat.
* So when monir is constructed:
  + Only **one copy** of rahat is initialized.
  + Only one **"student"** is printed.
* Then constructors continue as usual:
  + Man
  + Good student
  + amr vhul hoyeche

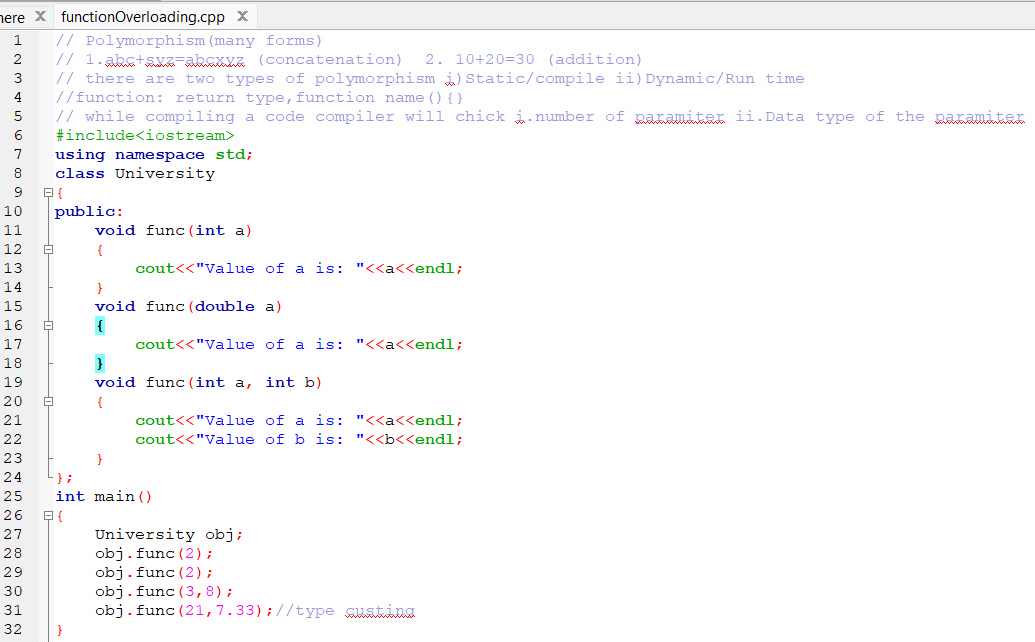
**LAB-14**

**Title of the problem: IMPLEMENT THE FUNCTION OVERLOADING IN OOP**

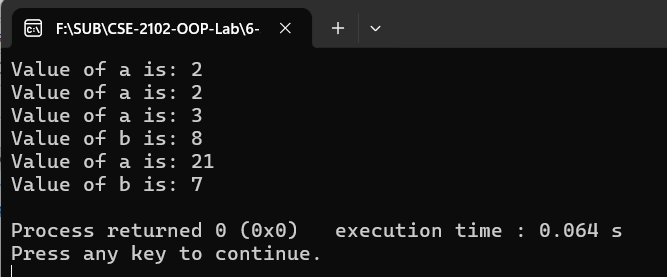
**Test case analysis:**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| obj.func(2) | Value of a is: 2 |
| obj.func(2) | Value of a is: 2 |
| obj.func(3,8) | Value of a is: 3  Value of b is: 8 |
| obj.func(21,7,33) | Value of a is: 21  Value of b is: 7(after type casting) |

**Coding:**

****

**Result:**

****

**Justification of the output:**

1. **obj.func(2)**
   * Matches void func(int a)
   * Output: Value of a is: 2
2. **obj.func(3, 8)**
   * Matches void func(int a, int b)
   * Output:

csharp

CopyEdit

Value of a is: 3

Value of b is: 8

1. **obj.func(21, 7.33)**
   * There’s **no exact match** for (int, double)
   * **Implicit type casting** occurs: 7.33 → 7 (converted to int)
   * So, func(int a, int b) is called
   * Output:

csharp

CopyEdit

Value of a is: 21

Value of b is: 7

**LAB-15**

**Title of the problem: IMPLEMENT THE FUNCTION OVERRIDING IN OOP.**

**Test case analysis:**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
|  |  |
|  |  |
|  |  |

**Coding:**

**Result:**

**Justification of the output:**