**SECURIN Assessment Solution**

**Name: CHIRAG VARSHNEY**

**Roll No: S20200020254**

**Github Repo Link:** [**Asessment\_Solution**](https://github.com/phoenix1406/Securin)

Ques 01 : **Problem statement:**

You are given two six-sided dice, Die A and Die B, each with faces numbered from 1 to

6.

You can only roll both the dice together & your turn is guided by the obtained sum.

Example: Die A = 6, Die B = 3. Sum = 6 + 3 = 9

You may represent Dice as an Array or Array-like structure.

Die A = [1, 2, 3, 4, 5, 6] where the indices represent the 6 faces of the die & the value on

each face.

**PART A:**

1. How many total combinations are possible? Show the math along with the code!

* Logic To Problem Statement:

We have two dice, A and B, each with values [1, 2, 3, 4, 5, 6].We want to find the total number of combinations when rolling both dice.Each die has 6 faces, so there are 6 possible outcomes for each die and for each particular value on one dice , i will get 6 different combinations for my second dice since i have two dices so total combinations will be 6\*6 = 36 in total.

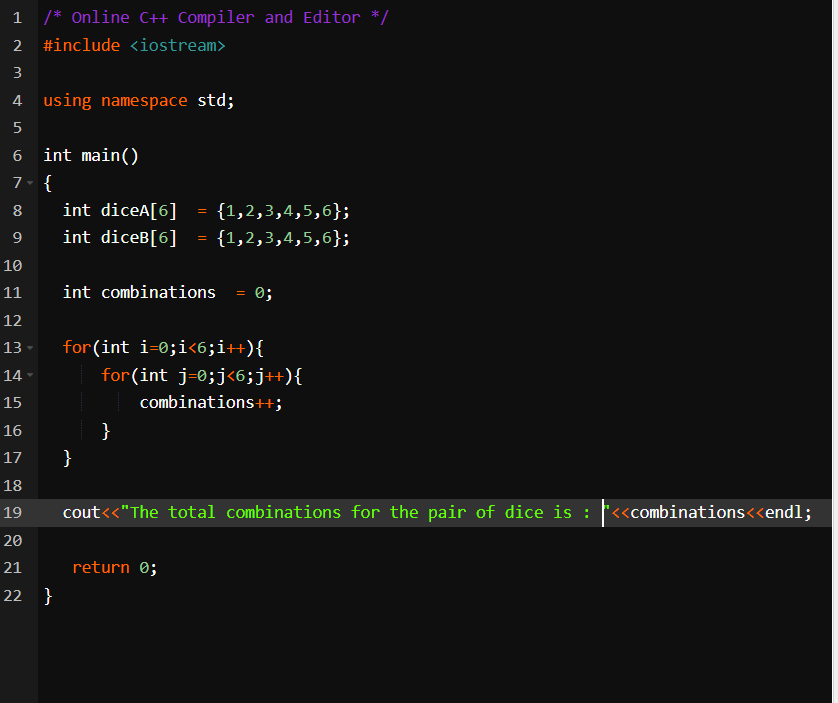
* Code and Output Screenshot:

1.By using 2 for loops to count all the possibilities: -

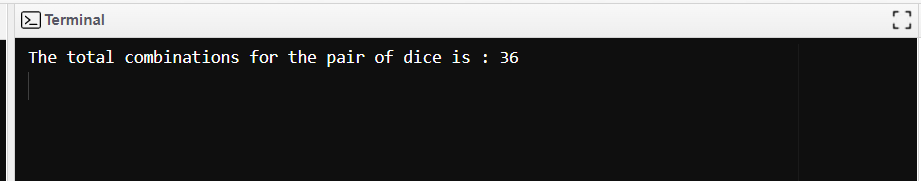
We can use two loops to iterate over both the dice and then we count

total combinations .

**Code Screenshot**

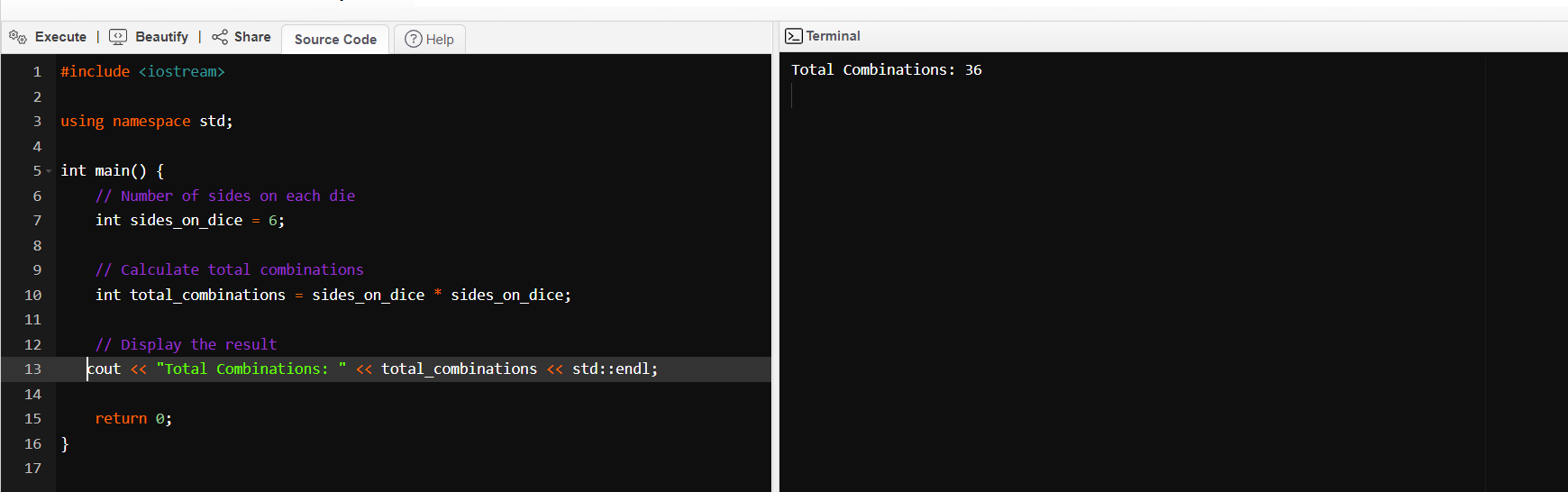


Output Screenshot



2.This can also be done by squaring the number of possibilities of a

Single dice as they have both equal no of cases or sides we say that 6.

Code and output screenshot :

* Solution Approach:  
  In summary, this C++ program calculates and displays the total combinations when rolling two dice with six sides each. The logic involves multiplying the number of sides on each die to find the total number of combinations.Or in the other logic we create two arrays of size 6 then for each combination on die A w e iterate it through 6 different combinations for die B using for loop and i increment my combinations variable to find total combinations and displayed it on terminal.

2. Calculate and display the distribution of all possible combinations that can be

obtained when rolling both Die A and Die B together. Show the math along with

the code.

* Logic To Problem Statement

To calculate the distribution of all possible combinations when rolling both Die A and Die B together, we need to consider each possible pair of outcomes and count their occurrences. The logic involves iterating through all combinations and keeping track of their frequencies.

Mathematical Approach:

We have two dice, each with 6 faces.

For each face of Die A, we can pair it with each face of Die B.

The distribution is the count of each unique sum of the pairs.

Mathematically: Distribution of Sum (S) = Count of (A, B) pairs with sum S.

Distribution of Sum:

Sum 2: 1 combinations

Sum 3: 2 combinations

Sum 4: 3 combinations

Sum 5: 4 combinations

Sum 6: 5 combinations

Sum 7: 6 combinations

Sum 8: 5 combinations

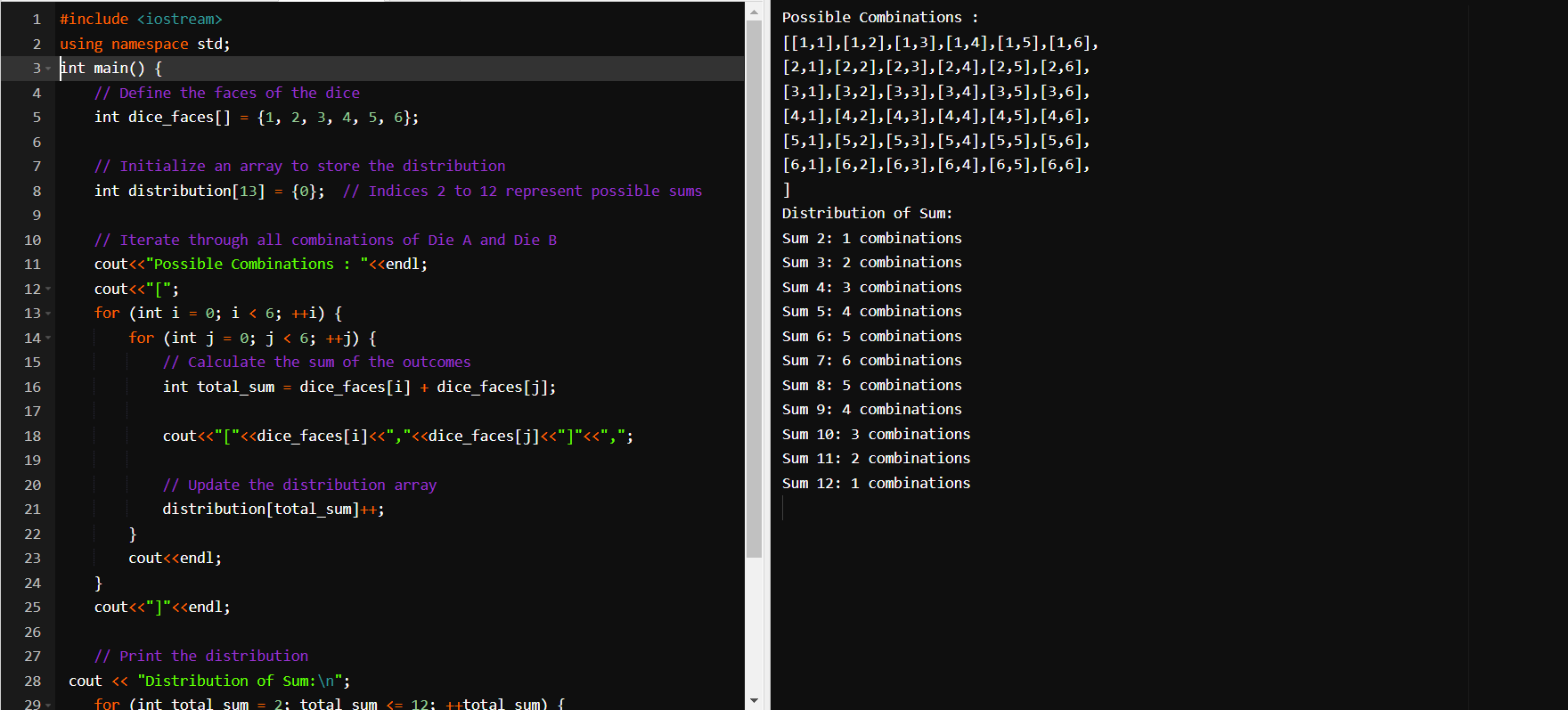
Sum 9: 4 combinations

Sum 10: 3 combinations

Sum 11: 2 combinations

Sum 12: 1 combinations

* Code and output Screenshot



* Solution Approach :

The provided C++ code generates and displays all possible combinations of two six-sided dice, along with the distribution of their sums.The array dice\_faces represents the faces of a six-sided die.

An array named distribution is initialized to store the distribution of sums. Indices 2 to 12 represent the possible sums when rolling two six-sided dice.The nested loops iterate through all combinations of Die A and Die B, calculating the sum of the outcomes for each pair. The distribution array is updated accordingly.In summary, the code provides a clear visualization of all possible combinations and their distribution when rolling two six-sided dice.

3) Calculate the Probability of all Possible Sums occurring among the number of combinations from (2).

Example: P(Sum = 2) = 1/X as there is only one combination possible to obtain

Sum = 2. Die A = Die B = 1.

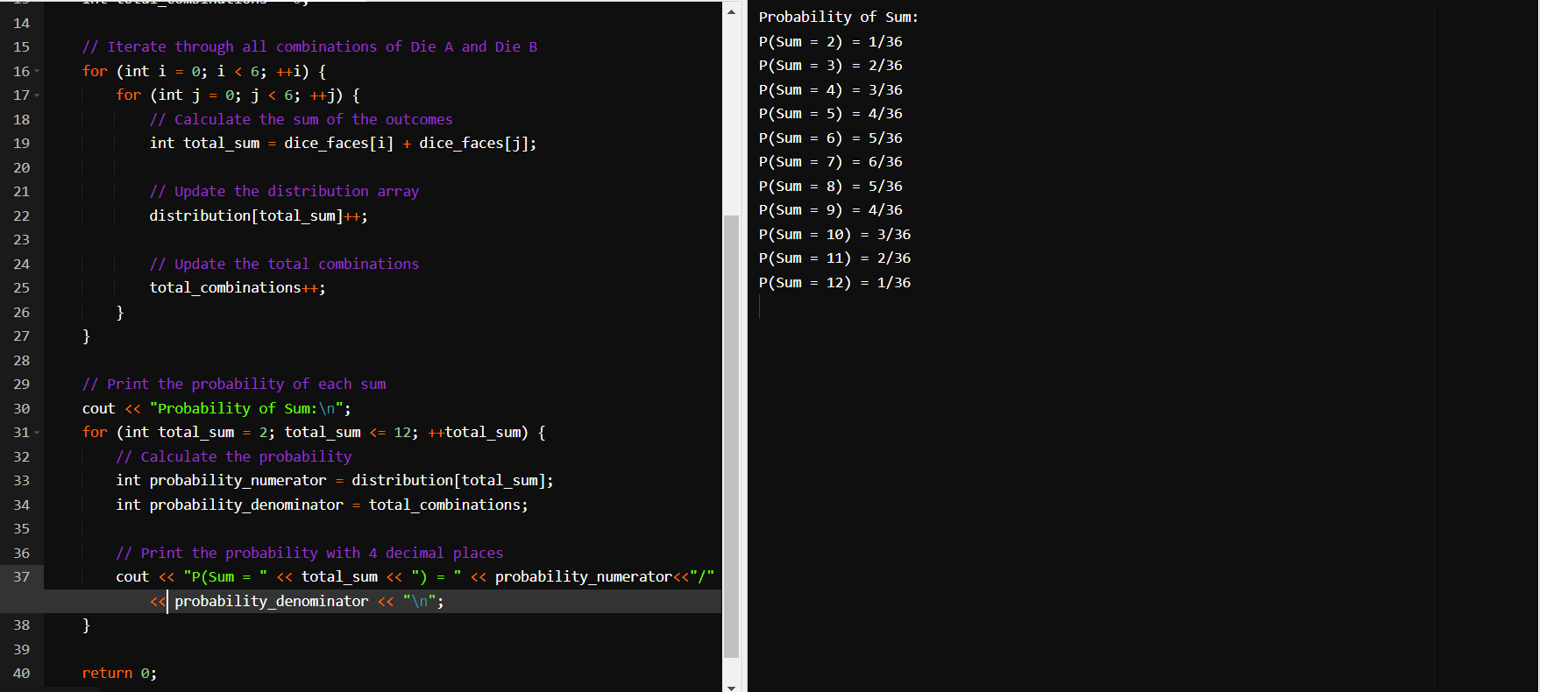
* Logic To Problem:

Certainly! The probability of a specific sum occurring is the ratio of the number of favorable outcomes (combinations resulting in that sum) to the total number of possible outcomes. The formula for probability (P) is given by:

P(Sum=X)=

Number of combinations resulting in Sum X / Total Number of combinations.

Code and Output Screenshot:



* Solution Approach :

Dice Faces: An array dice\_faces is defined to represent the faces of a six-sided die.

Distribution Array: An array distribution is initialized to store the frequency of each possible sum (from 2 to 12).

Total Combinations: The variable total\_combinations is initialized to keep track of the total number of combinations when rolling two dice.

Nested Loop: Two nested loops iterate over all combinations of rolling two dice, calculating the sum of the outcomes.

Update Distribution: The distribution array is updated based on the calculated sum.

Update Total Combinations: The total\_combinations variable is updated.

Print Probability: After all combinations are considered, the program prints the probability of each sum by dividing the frequency of each sum by the total number of combinations.

Output Format: The output displays the probability of each sum with four decimal places.

**PART B:**

**Problem Statement :**

**Now comes the real challenge. You were happily spending a lazy afternoon playing**

**your board game with your dice when suddenly the mischievous Norse God Loki ( You**

**love Thor too much & Loki didn’t like that much ) appeared.**

**Loki dooms your dice for his fun removing all the “Spots” off the dice.**

**No problem! You have the tools to re-attach the “Spots” back on the Dice.**

**However, Loki has doomed your dice with the following conditions:**

**● Die A cannot have more than 4 Spots on a face.**

**● Die A may have multiple faces with the same number of spots.**

**● Die B can have as many spots on a face as necessary i.e. even more than 6.**

**But in order to play your game, the probability of obtaining the Sums must remain the**

**same!**

**So if you could only roll P(Sum = 2) = 1/X, the new dice must have the spots reattached**

**such that those probabilities are not changed.**

**Input:**

**● Die\_A = [1, 2, 3, 4, 5, 6] & Die B = Die\_A = [1, 2, 3, 4, 5, 6]**

**Output:**

**● A Transform Function undoom\_dice that takes (Die\_A, Die\_B) as input &**

**outputs New\_Die\_A = [?, ?, ?, ?, ?, ?],New\_Die\_B = [?, ?,**

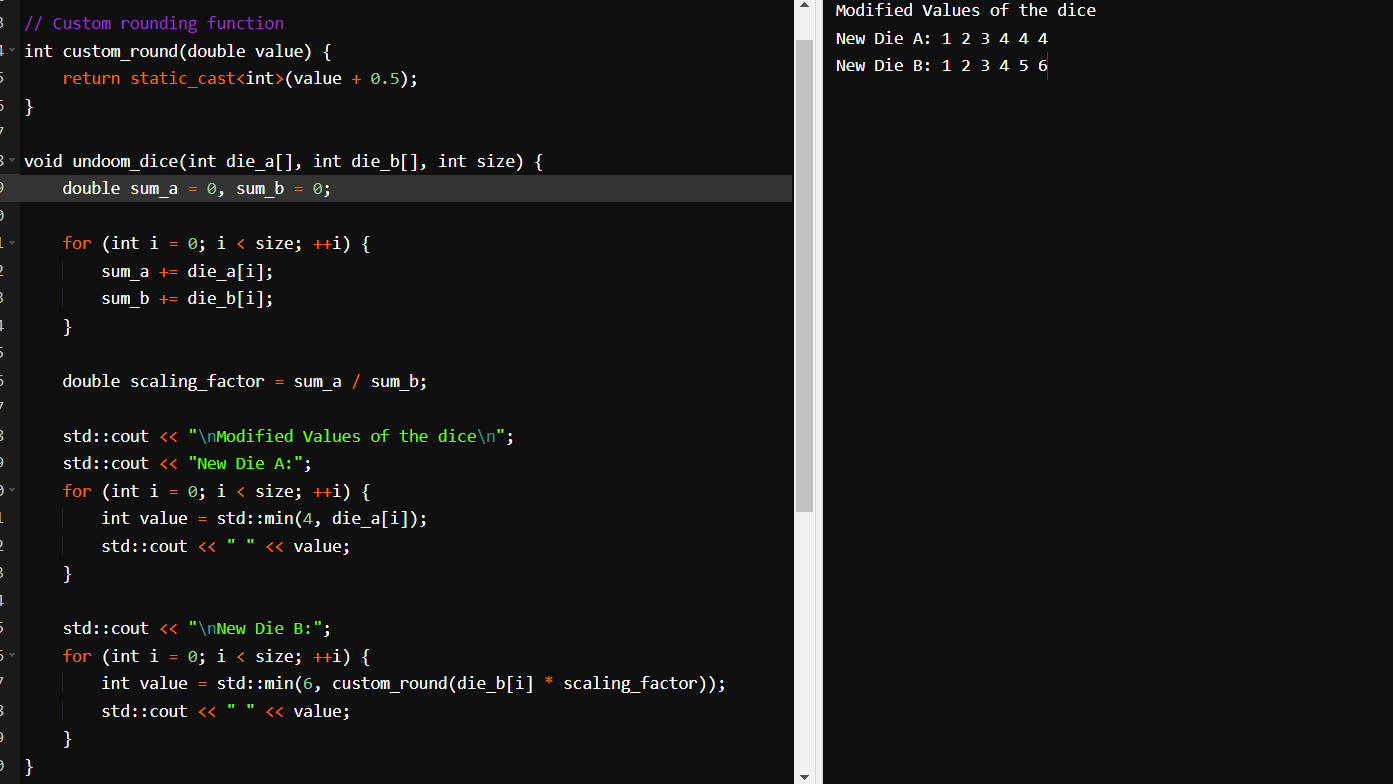
**?, ?, ?, ?] where,**

**● No New\_Die A[x] > 4**

Logic To Problem:

The idea is to design and modify the values of two sets of dice (die\_a and die\_b) based on a scaling factor derived from the sums of their initial values. The goal is to adjust the values of die\_b such that their sum matches the sum of values in die\_a. Additionally, some constraints are applied to the new values, ensuring that the modified values for each die are capped at certain limits (4 for die\_a and 6 for die\_b).The scaling factor is used to adjust the values of the second set of dice (die\_b) in such a way that the sum of the modified values matches the sum of the original values of the first set of dice (die\_a). This scaling ensures that the two sets of dice have similar total values after modification.The idea is to maintain the proportionality of the original values in die\_b while adjusting their magnitudes to ensure the total sum matches that of die\_a.In this way the probabilities of their sum in original dice is maintained with the new dice which we get

Code and Output Screenshot:



Solution Approach:

The code is implementing a custom dice manipulation function (undoom\_dice) where it modifies the values of two sets of dice (die\_a and die\_b). The modification involves scaling the values of die\_b based on the ratio of the sum of values in die\_a to the sum of values in die\_b.The program defines a custom rounding function custom\_round.

In the main function, two arrays of integers die\_a and die\_b are declared and initialized.

The undoom\_dice function is called with die\_a and die\_b arrays.It calculates the sum of values in die\_a and die\_b.

It computes a scaling factor based on the ratio of the sum of die\_a to the sum of die\_b.

It then prints modified values of the dice:

For die\_a, it prints the values, clamped to a maximum of 4.

For die\_b, it scales each value using the scaling factor, rounds it using custom\_round, and clamps it to a maximum of 6.

**Note:** All cpp solutions are compiled in gnu C++ compiler and for part A a combine code has been attached for reference and similarly for part B a code has been attached for reference in the github repo.

THANK YOU

END OF ASSIGNMENT

​