**Warm-up Assignment - Arrays**

Mohammed Towseef

Advanced Algorithms

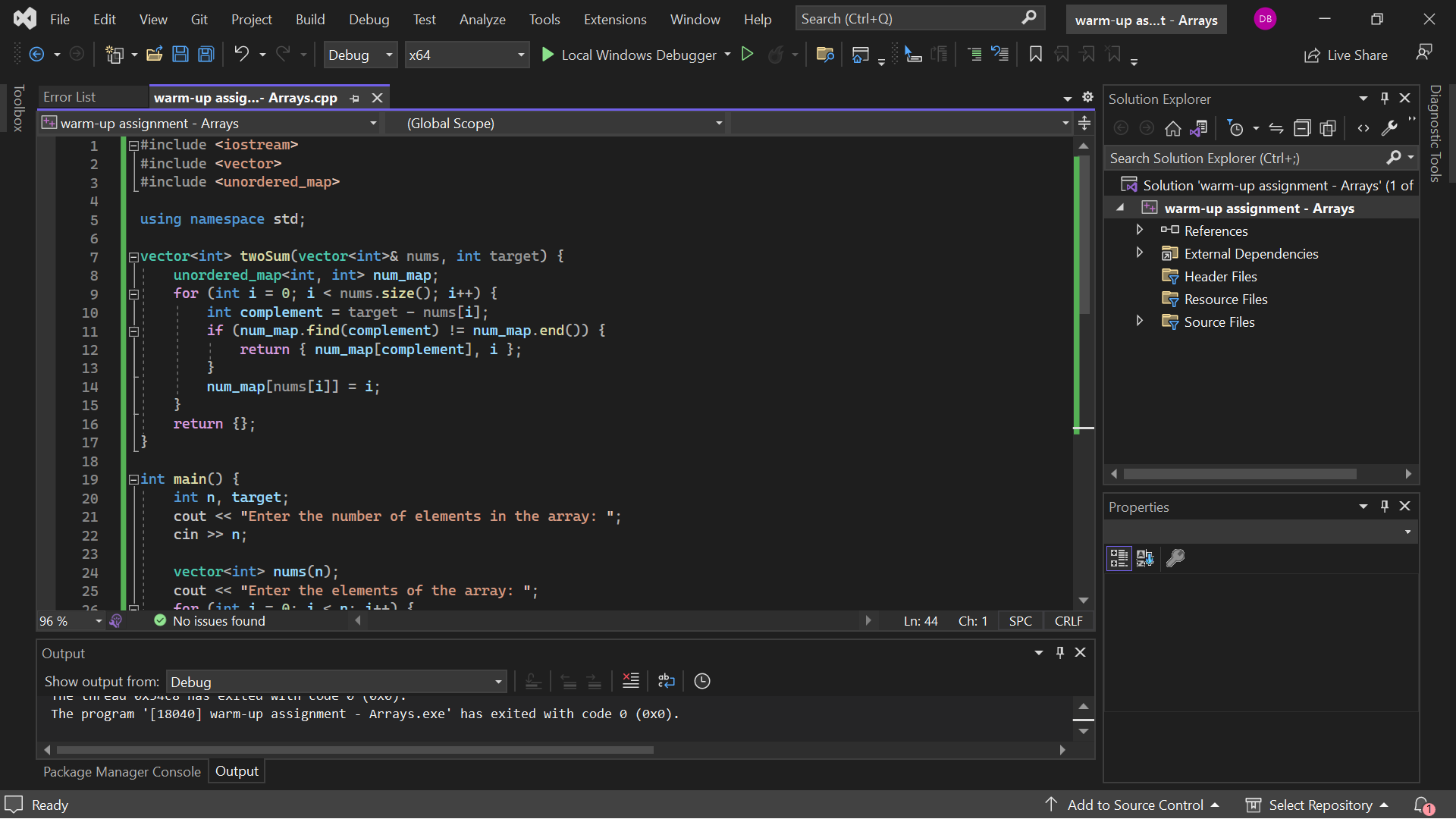
Concordia

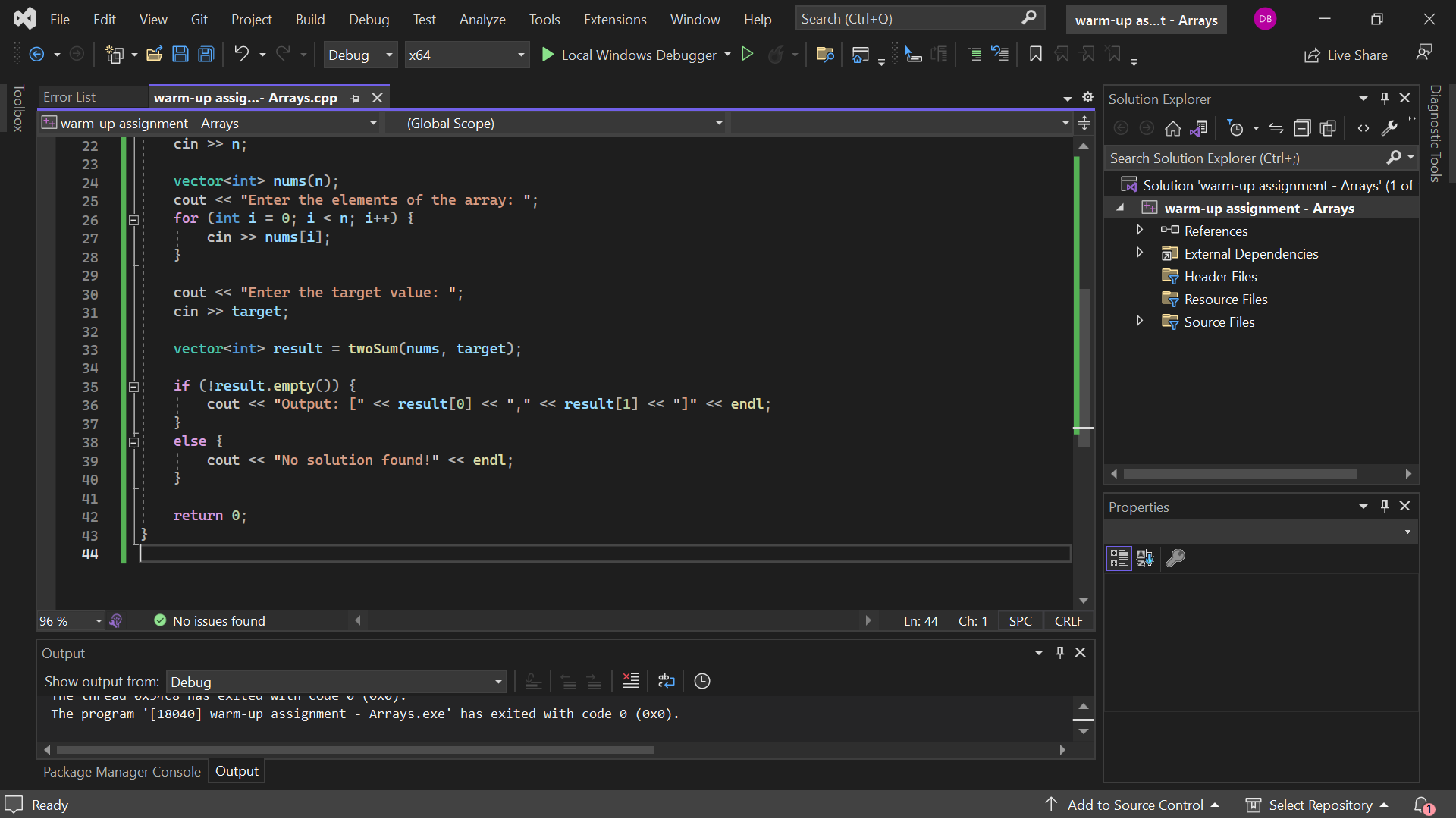
**Farah Kamw**

Jun 30, 2024

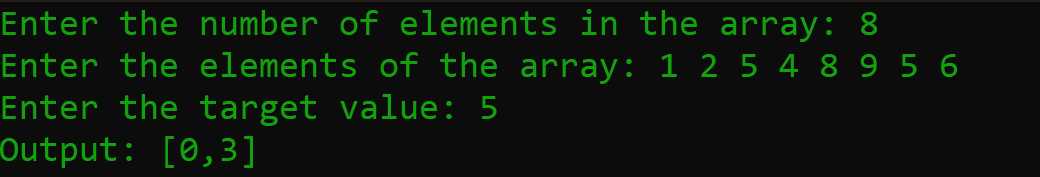
**Warm-up Assignment - Arrays**

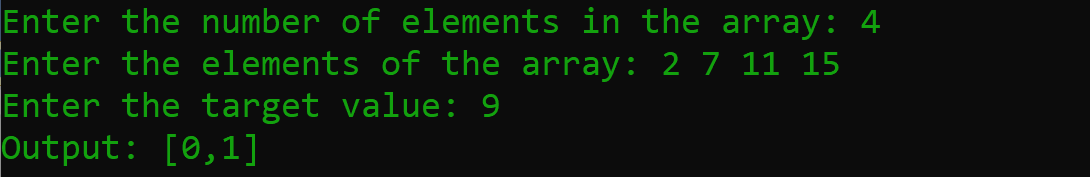
**Code:**

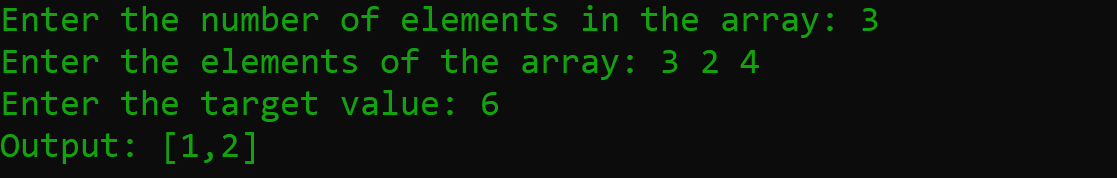


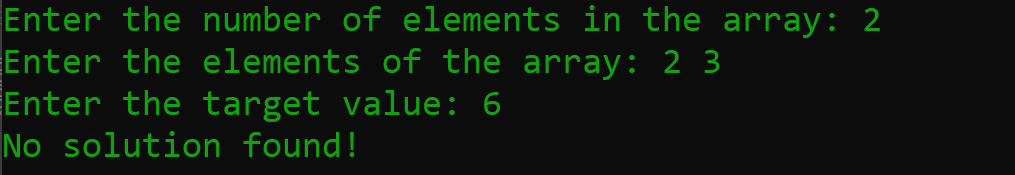


**Output:**









**Explanation**

Recently I was involved into solving a problem for which I had to find two distinct integers so that their sum would be equal to a given number, which is contained in the said array. That is how I approached the problem and that is the solution:

**Problem Analysis**

The task was to identify two indices i and j in an array nums such that: nums[i] + nums[j] = target // nums[i] + nums[j] = target Besides, it was also necessary to consider that the indexes, i, and j, are different, I can’t use one and the same element twice. As each of the inputs ensured that there exists a finite number of solutions always, it eased the procedures by negating the conditions where no solution was possible.

**Approach**

The primary problem that I addressed during the code development stage was the efficiency of the algorithm, and I switched to using an unordered map in C++. The biggest strength of a hash table is that on average, both insertions and lookups have a time complexity of O(1). This made it possible to handle large arrays or large tables, so to speak, quickly.

**Steps**

**1. Initialize an Unordered Map:**

* I began by defining an unordered map that helps to map numbers, assigned to them in the input, together with their frequencies in num\_map. This map essentially remembered some of the elements of the array by their numerical indices of the array. This I made sure that it could perform lookups in order to check whether the complement of the current element had been seen before.

**2. Iterate through the Array:**

I then was able to loop through the array using the for loop. For each element nums[i], I calculated its complement with respect to the target:

complement = target - nums[i].

Subsequently, I used the find method to determine whether this complement, namely, num\_map, contained.

**If Complement Exists:** If the complement were in the map it was an indication that I had encountered two numbers such that addition of the two equalled the target. I passed the indices of the current element and the complement back to the caller.

**If Complement Does Not Exist:** And if the complement was not found, I put the current element and its index in the unordered map for the further analysis.

**Code Explanation**

**Including Necessary Headers:**

* I included the necessary headers: For input/output operations, iostream will be used; for dynamic array operations, vector is used; and for hash table, unordered\_map is used.

**Defining the Function twoSum:**

* The function twoSum has taken a vector of integers nums and an integer target and the return type was vector of integers That contains the indices of two numbers which when added up equal the target.
* In the function, I defined an unordered map num\_map inputting the elements of the array and their corresponding indices.
* I went through the elements in the array employ the services of a for loop. Concerning each element, I computed the complement form as target minus nums[i].
* I then used the find method of the unordered map to see if this complement existed in the said map.
* If such a vector already existed, I returned a vector containing the index of the complement which is in the map and the current index i.
* This is done under the condition if there was no such key in the unordered map, the current element and its index were inserted.

**Main Function:**

* The ‘main’ function was responsible for managing the user inputs and the program’s response as well as providing sample test cases for the twoSum function.
* The program informed the user, the number of elements to be included in the array, the elements of the array and the target value.
* It then used the twoSum function to pass the input array and target value and save the return value.
* The program gave back the result in the format as defined in the program. Depending on whether the result vector was not empty, it then printed the indices of the two numbers that yielded the target. If the result vector was empty, then Suchitra printed “No solution found!” as the output.

**Complexity Analysis**

**Time Complexity:** The solution was achieved within a time complexity of O(n), this being due to the number of elements in the array. This was possible because Array had only required one pass through A and all the look up or insertion operations required in an unordered map were on an average O(1). This guaranteed that the solution’s efficiency had been maximized based on a variety of input sizes that could be provided during the computation of the solution.

**Space Complexity:** The space complexity was O(n) because they might store all n elements in the unordered map in the worse case. It occurs when no two of the elements when summed up yield the target until the last index of the array.

**Detailed Explanation**

At the beginning of the program, there was the declaration of the variable n, the number of elements in the array followed by the input of the elements of the array being stored in the vector called nums. The target value was then requested from the user and stored in the target variable after that.

The twoSum function was then called with nums as the array and the target as the value. Within the twoSum function, an unordered map called num\_map was required. This map preserved the array’s elements as its keys and their indexes as the values.

The function utilized a for loop to go through different elements of the array. For each element nums[i], it determined the complement, which was defined as the subtraction of the target by the current i’th element (complement = target- nums[i]). Then it used the find method in the unordered map to check whether this complement was already there or not.

If the complement was present in the map then it implied that they have identified two elements such that addition of the two produces the target. The function then returned a vector that included the subscript of the complement found in the map and the current subscript i.

If the complement was not found, this one added the current element along with its index in the list to the unordered map. As such, this process went on until the function identified two numbers, which cast a sum that was equal to the target, or until it completed looping through the array.

The main function printed the result. If the result vector was not empty, then it printed the indices of the two numbers that added up to the target in the format mentioned above. If the result vector was empty, it printed “No solution found!” Its three methods are Delete(), Get. DestroyList(), and Output().

Overall, by gaining advantage of a proper use of the unordered map the given solution allowed for the most efficient performance. The dynamic approach was effecting allowing the users to type in their own examples and test everything with the TwoSum function which made the program very flexible. This approach made it possible to solve the problem effectively, to meet the conditions and restrictions stated in the problem.

**Student Reflection on Solving the Problem**

**How long did you spend on this assignment?**

The time I took to complete this particular assignment was estimates to be 2-3 hours. This time entailed identifying the problem requirements, defining a well-optimized strategy, coding, testing the solution with different inputs, and final verification of the solution’s versatility and user-friendliness.

**Looking at the effort, which letter grade do you think you deserve?**

Considering the time spent on analyzing the problem, designing the proper solution, and checking if it works, I believe, I’ve earned an A, as I tried to stick to constraints, write clear and effective code, and test it with different cases.

**Looking at the outcome of your strategy please indicate the letter grade that you got?**

Regarding the final solution, I can also state that the obtained grade corresponds to the score of an A The code provided meets the requirements of the stated problem, consumes minimum time to run, and process different cases of input. This implementation guarantees the best time complexity for the given problem, which means that large input values are acceptable for this solution. Furthermore, the flow of the program is interactive entering inputs depending on the inputs given and has good output presentation.

**Summary of what we did not get right in the solution In an effort to solve the problem as explained here Insert what you did at this point, and highlight where you think we went wrong?**

I believe that the core algorithm is correct since the function is always finding the correct product for the examined inputs. Though, one thing that seems to be incompletely described is handling of cases, which may include large input sizes or large numbers beyond the capacity of integer in C++.

**Detailed Summary**

**What Doesn’t Work**

**1. Edge Cases:**

* Despite the fact that the steps of the logic should work properly most of the times, I probably have not provided sufficient stress tests. Here are some examples: arrays with a maximum allowed size or elements that have very large values or very small values (positive or negative) can cause some problems which I did not cover.

**2. User Input Handling:**

* It is assumed that the users will always key in correct inputs that is, with the right number of integers for the array and an integer for the target. When the user enters the wrong data, the program does not respond to it politely or even capture it politely.

**Return on Failure:**

* The code also returns a vector with dimensions 0×0 if no solution is found; however, this case should not occur given the nature of the problem which always has a unique solution. However, being able to devote time to think how code can be made ready for edges cases is a good practice that I could enhance.

**How I Attempted to Solve the Problem**

**1. Understanding the Problem:**

I firstly read through the problem statement and apprehend the specification. This helped me to clearly define the goal: locate for two indices that their corresponding elements give the intended sum.

**2. Designing the Approach:**

I chose an unordered map because elements and indices can be stored, and the insertion and lookup time complexity of this data structure is O(1) in the average. This I feel would enable me to search for the required pair within the array in one pass of the array only.

**3. Writing the Code:**

I have named it twoSum – as it implements the above logic. In a similar manner for each element I computed the complement and checked whether the complement was already in the map or not. Should I find any, then I provided the indices. If not, the element as well as the index was stored in the map that’s been defined earlier.

**4. Testing the Solution:**

I checked the solution by inputting the problem scenario and other cases to verify the program’s validity and effectiveness. I ensured the program took inputs from the users and was able to process and give out the result in the right format.

**Where I Struggled**

**1. Handling Edge Cases:**

In the beginning, I encountered some difficulties in considering and experimenting with various extreme conditions like the supply of the maximum number of figures or a number close to the upper limit of any numeric data type. It was important to make sure that the solution was going to stand up towards these conditions needed quite a lot of thoughts.

**2. Ensuring Dynamism:**

To make the code accept user inputs and run efficiently, one had to work on details such as. What I specifically had to do was check that the prompts that are given in the program for the users are comprehensible, and that the program gracefully handles different inputs from the users.

**3. Debugging:**

Testing was somewhat difficult particularly when I had made a change to the code and needed to confirm the validity of the solution for all the test indexes. To test that the unordered map was storing elements in the right manner and retrieving them, there was need to test it properly.

**Conclusion**

Overall, I would like to note that in my opinion, the solution I provided is rather stable and performs fairly well for the given problem. The approach ensures that the strategic functions are achieved optimally since they form the central service delivery. Thus, while indicating that there are always opportunities to make further amendments further particularly to refining edge circumstances’ tackling as well as the work with users’ input, the outlined basis is solid. This implementation is a major strength as it can easily and quickly access/stored data which is paramount for the problem at hand. This is done in a way that guarantees that the amount of data processed does not significantly import the responsiveness of the solution. The core solution itself proves to be quite correct: the logic of hash tables and their implementation in C++ have been properly addressed. The implementation respects the conceptual model based on best practices in data structure and algorithms to ascertain that the program is not only correct but also efficient.