Two Sum in C++

```
#include <iostream>
#include <unordered_map>
#include <vector>

using namespace std;

vector<int> twoSum(vector<int>& nums, int target) {
    unordered_map<int, int> map; // Hash map to store number and its index
    vector<int> result;

    for (int i = 0; i < nums.size(); i++) {
        int complement = target - nums[i];

        if (map.find(complement) != map.end()) {
            result.push_back(map[complement]);
            result.push_back(i);
            return result;
        }

        map[nums[i]] = i;
    }

    throw invalid_argument("No two sum solution");
}

int main() {
    vector<int> nums1 = {2, 7, 11, 15};
    int target1 = 9;

    vector<int> nums2 = {3, 2, 4};
    int target2 = 6;

    vector<int> result1 = twoSum(nums1, target1);
    vector<int> result2 = twoSum(nums2, target2);

    cout << "Output for nums1: [" << result1[0] << ", " << result1[1] << "]" << endl;
    cout << "Output for nums2: [" << result2[0] << ", " << result2[1] << "]" << endl;

    return 0;
}
```

Explanation of Code:

The given code is solving the classic "Two Sum" problem. The task is to find two indices in an array where the values at those indices add up to a specific target sum.

Step-by-Step Breakdown:

1. **Input**:
   - nums1 = {2, 7, 11, 15}, target1 = 9
   - nums2 = {3, 2, 4}, target2 = 6
2. **Core Logic**:
   - **twoSum function**:
     - A hash map (unordered_map) is used to store each element in the array and its corresponding index.
     - The idea is to check, for each element nums[i], whether its complement (target - nums[i]) already exists in the map. If it does, we've found the solution.
     - If not, we store the element nums[i] along with its index in the map for future reference.
3. **Detailed Steps**:
   - For nums1 = {2, 7, 11, 15} and target1 = 9:
     1. We start iterating through the array.
     2. For i = 0 (value 2), we calculate the complement: 9 - 2 = 7. The map is empty, so we store 2 in the map with its index: map = {2: 0}.
     3. For i = 1 (value 7), we calculate the complement: 9 - 7 = 2. Since 2 is already in the map (at index 0), we've found the solution: indices [0, 1].
     4. The result [0, 1] is returned.
   - For nums2 = {3, 2, 4} and target2 = 6:
     1. We start iterating through the array.
     2. For i = 0 (value 3), we calculate the complement: 6 - 3 = 3. The map is empty, so we store 3 in the map with its index: map = {3: 0}.
     3. For i = 1 (value 2), we calculate the complement: 6 - 2 = 4. Since 4 is not in the

	map, we store 2 with its index: map = {3: 0, 2: 1}. 4. For i = 2 (value 4), we calculate the complement: 6 - 4 = 2. Since 2 is already in the map (at index 1), we've found the solution: indices [1, 2]. 5. The result [1, 2] is returned. **Output:** Output for nums1: [0, 1] Output for nums2: [1, 2]
Output:- Output for nums1: [0, 1] Output for nums2: [1, 2]	