Isomorphic Strings in C++

```
#include <iostream>
#include <string>
#include <unordered_map>

using namespace std;

bool iso(string s, string t) {
    if (s.length() != t.length()) {
        return false;
    }

    unordered_map<char, char> map1; // Maps characters from s to t
    unordered_map<char, bool> map2; // Tracks characters used in t

    for (int i = 0; i < s.length(); i++) {
        char ch1 = s[i];
        char ch2 = t[i];

        if (map1.count(ch1) > 0) { // If ch1 is already mapped
            if (map1[ch1] != ch2) { // Check if mapping is consistent
                return false;
            }
        } else { // ch1 has not been mapped yet
            if (map2.count(ch2) > 0) { // If ch2 is already mapped by another character in s
                return false;
            } else { // Create new mapping
                map1[ch1] = ch2;
                map2[ch2] = true;
            }
        }
    }

    return true;
}

int main() {
    string s1 = "abc";
    string s2 = "cad";
    cout << boolalpha << iso(s1, s2) << endl; // Output: true

    return 0;
}
```

Dry Run:

Input:

s1 = "abc"
s2 = "cad"

Step 1 - Check Length:

- First, we check if the lengths of s1 and s2 are the same. Both are of length 3, so we proceed.

Step 2 - Initialize Maps:

- map1 (for mapping characters of s1 to s2) is an empty map initially.
- map2 (to track characters already used in s2) is also an empty map initially.

Step 3 - Iterate Over the Strings:

Now we iterate over each character in s1 and s2 simultaneously:

1. **First iteration** (i = 0):
    - ch1 = s1[0] = 'a' and ch2 = s2[0] = 'c'
    - 'a' is not mapped yet, and 'c' is not used yet in map2.
    - So, we create a mapping 'a' -> 'c' in map1 and mark 'c' as used in map2.
2. **Second iteration** (i = 1):
    - ch1 = s1[1] = 'b' and ch2 = s2[1] = 'a'
    - 'b' is not mapped yet, and 'a' is not used yet in map2.
    - So, we create a mapping 'b' -> 'a' in map1 and mark 'a' as used in map2.
3. **Third iteration** (i = 2):
    - ch1 = s1[2] = 'c' and ch2 = s2[2] = 'd'
    - 'c' is not mapped yet, and 'd' is not used yet in map2.
    - So, we create a mapping 'c' -> 'd' in map1 and mark 'd' as used in map2.

Step 4 - Check Mapping Consistency:

After the loop ends, the mappings in map1 are:

map1 = { 'a' -> 'c', 'b' -> 'a', 'c' -> 'd' }

Since all characters in s1 have been mapped to distinct characters in s2, and no character in s2 has been mapped by more than one character from

	s1, the strings are isomorphic. **Final Output:** Since the mappings are valid and consistent, the function returns true.
Output: true	