LCS in C++

```cpp
#include <iostream>
#include <string>
#include <algorithm> // For std::max
using namespace std;

// Define maximum possible sizes for the strings
const int MAX_M = 100;
const int MAX_N = 100;

int LCS(const string& s1, const string& s2) {
    int m = s1.length();
    int n = s2.length();

    // Initialize DP table with zeros
    int dp[MAX_M + 1][MAX_N + 1] = {0};

    for (int i = m - 1; i >= 0; i--) {
        for (int j = n - 1; j >= 0; j--) {
            if (s1[i] == s2[j]) {
                dp[i][j] = 1 + dp[i + 1][j + 1];
            } else {
                dp[i][j] = max(dp[i + 1][j], dp[i][j + 1]);
            }
        }
    }

    return dp[0][0];
}

int main() {
    string s1 = "abcd";
    string s2 = "abbd";

    cout << LCS(s1, s2) << endl;

    return 0;
}
```

Dry Run:

Let's break down the execution of the code with the strings s1 = "abcd" and s2 = "abbd":

- **Step 1: Initializing the DP table**

  We initialize a DP table of size (5x5) (since s1.length() = 4 and s2.length() = 4, we add 1 for the zero-indexed table).

  int dp[5][5] = {0};

  Initial table:

  0 0 0 0 0
  0 0 0 0 0
  0 0 0 0 0
  0 0 0 0 0
  0 0 0 0 0

- **Step 2: Filling the DP table**

  We start filling the table from the bottom-right corner to the top-left (i.e., in reverse order of the strings).

  - Compare s1[3] = 'd' with s2[3] = 'd': They are equal, so dp[3][3] = 1 + dp[4][4] = 1 + 0 = 1.
  - Compare s1[3] = 'd' with s2[2] = 'b': They are different, so dp[3][2] = max(dp[4][2], dp[3][3]) = max(0, 1) = 1.
  - Compare s1[3] = 'd' with s2[1] = 'b': They are different, so dp[3][1] = max(dp[4][1], dp[3][2]) = max(0, 1) = 1.
  - Compare s1[3] = 'd' with s2[0] = 'a': They are different, so dp[3][0] = max(dp[4][0], dp[3][1]) = max(0, 1) = 1.

  Now, the table looks like:

  0 0 0 0 0
  0 0 0 0 0
  0 0 0 0 0
  0 1 1 1 0
  0 0 0 0 0

  - Compare s1[2] = 'c' with s2[3] = 'd': They are different, so dp[2][3] = max(dp[3][3], dp[2][4]) = max(1, 0) = 1.
  - Compare s1[2] = 'c' with s2[2] = 'b': They are different, so dp[2][2] = max(dp[3][2], dp[2][3]) = max(1, 1) = 1.

	- Compare s1[2] = 'c' with s2[1] = 'b': They are different, so dp[2][1] = max(dp[3][1], dp[2][2]) = max(1, 1) = 1. - Compare s1[2] = 'c' with s2[0] = 'a': They are different, so dp[2][0] = max(dp[3][0], dp[2][1]) = max(1, 1) = 1. Now, the table looks like: 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0 - Compare s1[1] = 'b' with s2[3] = 'd': They are different, so dp[1][3] = max(dp[2][3], dp[1][4]) = max(1, 0) = 1. - Compare s1[1] = 'b' with s2[2] = 'b': They are equal, so dp[1][2] = 1 + dp[2][3] = 1 + 1 = 2. - Compare s1[1] = 'b' with s2[1] = 'b': They are equal, so dp[1][1] = 1 + dp[2][2] = 1 + 1 = 2. - Compare s1[1] = 'b' with s2[0] = 'a': They are different, so dp[1][0] = max(dp[2][0], dp[1][1]) = max(1, 2) = 2. Now, the table looks like: 0 0 0 0 0 0 2 2 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0 - Compare s1[0] = 'a' with s2[3] = 'd': They are different, so dp[0][3] = max(dp[1][3], dp[0][4]) = max(1, 0) = 1. - Compare s1[0] = 'a' with s2[2] = 'b': They are different, so dp[0][2] = max(dp[1][2], dp[0][3]) = max(2, 1) = 2. - Compare s1[0] = 'a' with s2[1] = 'b': They are different, so dp[0][1] = max(dp[1][1], dp[0][2]) = max(2, 2) = 2. - Compare s1[0] = 'a' with s2[0] = 'a': They are equal, so dp[0][0] = 1 + dp[1][1] = 1 + 2 = 3. Final DP table: 3 2 2 1 0 2 2 2 1 0

	1 1 1 1 0 1 1 1 1 0 0 0 0 0 0 **Final Answer:** • The length of the Longest Common Subsequence (LCS) is dp[0][0] = 3.
Output:- 3	