LCS in C++

```
#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;
}
```

Step-by-Step Execution:

We initialize a **DP table** `dp[MAX_M+1][MAX_N+1]` with all zeros.

- **Strings Given:**

  ```
  s1 = "abcd" (m = 4)
  s2 = "abbd" (n = 4)
  ```

- **Table Size:** `dp[5][5]` (since we use indices `0` to `4` inclusive)

Dry Run Table (Index-Based Execution of DP Table)

Step	i	j	s1[i]	s2[j]	Match?	Formula Used	dp[i][j] Value
1	3	3	'd'	'd'	Yes	`dp[i][j] = 1 + dp[i+1][j+1]`	`dp[3][3] = 1 + 0 = 1`
2	3	2	'd'	'b'	No	`dp[i][j] = max(dp[i+1][j], dp[i][j+1])`	`dp[3][2] = max(0,1) = 1`
3	3	1	'd'	'b'	No	`dp[3][1] = max(0,1) = 1`	
4	3	0	'd'	'a'	No	`dp[3][0] = max(0,1) = 1`	
5	2	3	'c'	'd'	No	`dp[2][3] = max(1,0) = 1`	
6	2	2	'c'	'b'	No	`dp[2][2] = max(1,1) = 1`	
7	2	1	'c'	'b'	No	`dp[2][1] = max(1,1) = 1`	
8	2	0	'c'	'a'	No	`dp[2][0] = max(1,1) = 1`	
9	1	3	'b'	'd'	No	`dp[1][3] = max(1,0) = 1`	
10	1	2	'b'	'b'	Yes	`dp[1][2] = 1 + dp[2][3] = 1 + 1 = 2`	
11	1	1	'b'	'b'	Yes	`dp[1][1] = 1 + dp[2][2] = 1 + 1 = 2`	
12	1	0	'b'	'a'	No	`dp[1][0] = max(1,2) = 2`	
13	0	3	'a'	'd'	No	`dp[0][3] = max(1,0) = 1`	
14	0	2	'a'	'b'	No	`dp[0][2] = max(2,1) = 2`	
15	0	1	'a'	'b'	No	`dp[0][1] = max(2,2) = 2`	
16	0	0	'a'	'a'	Yes	`dp[0][0] = 1 + dp[1][1] = 1 + 2 = 3`	

Final DP Table After Execution

	a	b	b	d
a	3	2	2	1
b	2	2	2	1
c	1	1	1	1
d	1	1	1	1

Final Output

```
LCS("abcd", "abbd") = 3
```

The longest common subsequence is `"abd"` (of length **3**).

Output:-
3