Longest Palindromic subseq In C++

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int LongestPalindromicSubsequence(string str) {
    int n = str.length();
    //vector<vector<int>> dp(n, vector<int>(n, 0));
    int dp[n][n]={0};

    for (int g = 0; g < n; g++) {
        for (int i = 0, j = g; j < n; i++, j++) {
            if (g == 0) {
                dp[i][j] = 1;
            } else if (g == 1) {
                dp[i][j] = (str[i] == str[j]) ? 2 : 1;
            } else {
                if (str[i] == str[j]) {
                    dp[i][j] = 2 + dp[i + 1][j - 1];
                } else {
                    dp[i][j] = max(dp[i][j - 1], dp[i + 1][j]);
                }
            }
        }
    }

    return dp[0][n - 1];
}

int main() {
    string str = "abccba";

    int longestPalSubseqLen = LongestPalindromicSubsequence(str);
    cout << longestPalSubseqLen << endl;

    return 0;
}
```

Input:

str = "abccba", n = 6

Initialization:

1. Create a **DP table** (dp[i][j]) of size 6×66 \times 66×6 initialized to 0.
2. **Base case**: Fill diagonal elements (g = 0) because every single character is a palindrome of length 1.

Initial DP Table (after g = 0):

i\j	**a**	**b**	**c**	**c**	**b**	**a**
a	1	0	0	0	0	0
b	0	1	0	0	0	0
c	0	0	1	0	0	0
c	0	0	0	1	0	0
b	0	0	0	0	1	0
a	0	0	0	0	0	1

Iterate Over Gaps:

Gap = 1 (g = 1):

- Compare adjacent characters:
  - If str[i] == str[j], then dp[i][j] = 2.
  - Else, dp[i][j] = 1.

i\j	**a**	**b**	**c**	**c**	**b**	**a**
a	1	1	0	0	0	0
b	0	1	1	0	0	0
c	0	0	1	2	0	0
c	0	0	0	1	1	0
b	0	0	0	0	1	1
a	0	0	0	0	0	1

Gap = 2 (g = 2):

- For substrings of length 3 (str[i...j]):
  - If str[i] == str[j]: dp[i][j]=2+dp[i+1][j−1]dp[i][j] = 2 + dp[i+1][j-1]dp[i][j]=2+dp[i+1][j−1]

- Else: dp[i][j]=max(dp[i][j−1],dp[i+1][j])dp[i][j] = \max(dp[i][j-1], dp[i+1][j])dp[i][j]=max(dp[i][j−1],dp[i+1][j])

i\j	**a**	**b**	**c**	**c**	**b**	**a**
a	1	1	1	0	0	0
b	0	1	1	2	0	0
c	0	0	1	2	2	0
c	0	0	0	1	1	2
b	0	0	0	0	1	1
a	0	0	0	0	0	1

Gap = 3 (g = 3):

- For substrings of length 4:
  - Use the same recurrence relation.

i\j	**a**	**b**	**c**	**c**	**b**	**a**
a	1	1	1	2	0	0
b	0	1	1	2	4	0
c	0	0	1	2	2	0
c	0	0	0	1	1	2
b	0	0	0	0	1	1
a	0	0	0	0	0	1

Gap = 4 (g = 4):

i\j	**a**	**b**	**c**	**c**	**b**	**a**
a	1	1	1	2	4	0
b	0	1	1	2	4	4
c	0	0	1	2	2	0
c	0	0	0	1	1	2
b	0	0	0	0	1	1
a	0	0	0	0	0	1

Gap = 5 (g = 5):

i\j	**a**	**b**	**c**	**c**	**b**	**a**
a	1	1	1	2	4	6
b	0	1	1	2	4	4
c	0	0	1	2	2	0
c	0	0	0	1	1	2
b	0	0	0	0	1	1
a	0	0	0	0	0	1

	Final Result: • dp[0][n-1] = dp[0][5] = 6.
Output:- 6	