Count Palindromic Subsequence C++

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

int countPalindromicSubseq(const string& str) {
    int n = str.length();
    int dp[n][n] = {0};  // Initialize the 2D array

    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] = dp[i][j - 1] + dp[i + 1][j] + 1;
                } else {
                    dp[i][j] = dp[i][j - 1] + dp[i + 1][j] - dp[i +
1][j - 1];
                }
            }
        }
    }

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

int main() {
    string str = "abccbc";
    cout << countPalindromicSubseq(str) << endl;
    return 0;
}
```

Initial Setup:

- str = "abccbc"
- n = 6 (length of the string)
- dp is a 2D array where dp[i][j] represents the number of palindromic subsequences in the substring str[i...j].

Step-by-Step Execution:

Initialize dp:

- For g = 0 (single character substrings), we initialize dp[i][i] = 1, since any single character is a palindrome by itself.
- For g = 1 (two-character substrings), we initialize dp[i][i+1] = 2 if the characters are the same, otherwise dp[i][i+1] = 1.

Loop through all values of g (gap between i and j):

For each g, we calculate dp[i][j] where j = i + g and i is the starting index.

Iteration 1: g = 0 (substrings of length 1)

We go through each character and set dp[i][i] = 1 (each character is a palindrome of length 1).

- dp[0][0] = 1 (for 'a')
- dp[1][1] = 1 (for 'b')
- dp[2][2] = 1 (for 'c')
- dp[3][3] = 1 (for 'c')
- dp[4][4] = 1 (for 'b')
- dp[5][5] = 1 (for 'c')

Iteration 2: g = 1 (substrings of length 2)

We check pairs of consecutive characters. If they are the same, dp[i][i+1] = 2, otherwise dp[i][i+1] = 1.

- dp[0][1] = 1 (for "ab" as 'a' != 'b')
- dp[1][2] = 1 (for "bc" as 'b' != 'c')
- dp[2][3] = 2 (for "cc" as 'c' == 'c')
- dp[3][4] = 1 (for "cb" as 'c' != 'b')
- dp[4][5] = 1 (for "bc" as 'b' != 'c')

Iteration 3: g = 2 (substrings of length 3)

Now we look at substrings of length 3 and calculate dp[i][j] for each.

- dp[0][2] = 2 (for "abc" as 'a' != 'c', using the formula dp[0][1] + dp[1][2] - dp[1][1])
- dp[1][3] = 3 (for "bcc" as 'b' != 'c', using the formula dp[1][2] + dp[2][3] - dp[2][2])

	• dp[2][4] = 3 (for "ccc" as 'c' == 'c', using the formula dp[2][3] + dp[3][4] + 1) • dp[3][5] = 3 (for "cbc" as 'c' == 'c', using the formula dp[3][4] + dp[4][5] + 1) **Iteration 4: g = 3 (substrings of length 4)** • dp[0][3] = 3 (for "abcc" as 'a' != 'c', using the formula dp[0][2] + dp[1][3] - dp[1][2]) • dp[1][4] = 4 (for "bccb" as 'b' == 'b', using the formula dp[1][3] + dp[2][4] + 1) • dp[2][5] = 6 (for "ccbc" as 'c' != 'c', using the formula dp[2][4] + dp[3][5] - dp[3][4]) **Iteration 5: g = 4 (substrings of length 5)** • dp[0][4] = 5 (for "abccb" as 'a' != 'b', using the formula dp[0][3] + dp[1][4] - dp[1][3]) • dp[1][5] = 7 (for "bccbc" as 'b' != 'c', using the formula dp[1][4] + dp[2][5] - dp[2][4]) **Iteration 6: g = 5 (substrings of length 6)** • dp[0][5] = 9 (for "abccbc" as 'a' != 'c', using the formula dp[0][4] + dp[1][5] - dp[1][4]) **Final Output:** The result stored in dp[0][5] (which is the value representing the number of palindromic subsequences in the entire string "abccbc") is 9
Output:- 9	