2.Medium\06.Longest_palindromic_substring.cpp

```cpp
/*
Question:-
Given a string `s`, the task is to find the longest palindromic substring in `s`.

Example 1:
Input: s = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.

Example 2:
Input: s = "cbbd"
Output: "bb"

Approach:
1. We define a helper function `expandFromCenter` that takes a string `s`, and two indices `start` and `end` as input.
2. The function expands from the center and checks if the substring from `start` to `end` is a palindrome.
3. If the length of the current palindrome is greater than the maximum length seen so far (`maxLen`), we update the maximum length and the corresponding start and end indices (`ans_start` and `ans_end`).
4. We iterate over each character of the string `s` and consider it as a potential center for the palindrome.
5. We call `expandFromCenter` twice for each character - once for considering odd-length palindromes and once for even-length palindromes.
6. Finally, we return the substring of `s` that corresponds to the longest palindromic substring.

Code:*/

void expandFromCenter(string s, int start, int end, int& ans_start, int& ans_end, int& maxLen) {
    while (start >= 0 && end < s.size() && s[start] == s[end]) {
        if (end - start + 1 > maxLen) {
            ans_start = start;
            ans_end = end;
            maxLen = end - start + 1;
        }
        start--;
        end++;
    }
}

string longestPalindrome(string s) {
    string ans = "";
    int maxLen = 0, ans_start = -1, ans_end = -1;
    for (int i = 0; i < s.size(); i++) {
        // For odd length palindromes
        expandFromCenter(s, i, i, ans_start, ans_end, maxLen);
        // For even length palindromes
        expandFromCenter(s, i - 1, i, ans_start, ans_end, maxLen);
    }
    return (maxLen == 0) ? "" : s.substr(ans_start, ans_end - ans_start + 1);
}

/*
Time Complexity: O(n^2), where n is the length of the input string `s`. The nested loops iterate over all possible pairs of indices.
Space Complexity: O(1), as we are using a constant amount of extra space.
*/

```