|  |  |
| --- | --- |
| **Longest Palindromic substring In C++** | |
| #include <iostream>  #include <string>  using namespace std;  int LongestPalindromicSubstring(string str) {      int n = str.length();      bool dp[n][n];      int len = 0;      // Initialize dp array      for (int i = 0; i < n; i++) {          dp[i][i] = true;      }      // Check for substrings of length 2      for (int i = 0; i < n - 1; i++) {          if (str[i] == str[i + 1]) {              dp[i][i + 1] = true;              len = 2; // Update length of longest palindromic substring          } else {              dp[i][i + 1] = false;          }      }      // Check for substrings of length > 2      for (int g = 2; g < n; g++) {          for (int i = 0, j = g; j < n; i++, j++) {              if (str[i] == str[j] && dp[i + 1][j - 1]) {                  dp[i][j] = true;                  len = g + 1; // Update length of longest palindromic substring              } else {                  dp[i][j] = false;              }          }      }      return len;  }  int main() {      string str = "abccbc";      int longestPalSubstrLen = LongestPalindromicSubstring(str);      cout << longestPalSubstrLen << endl;      return 0;  } | ****Step-by-Step Dry Run********Step 1: Initialize DP Table (****g = 0****)**** Each **single character** is a palindrome (dp[i][i] = true).   |  | **a** | **b** | **c** | **c** | **b** | **c** | | --- | --- | --- | --- | --- | --- | --- | | **a** | ✅ |  |  |  |  |  | | **b** |  | ✅ |  |  |  |  | | **c** |  |  | ✅ |  |  |  | | **c** |  |  |  | ✅ |  |  | | **b** |  |  |  |  | ✅ |  | | **c** |  |  |  |  |  | ✅ |   **Longest palindrome so far: len = 1** (since all single characters are palindromes). ****Step 2: Substrings of Length 2 (****g = 1****)**** We check adjacent characters str[i] == str[i+1].   |  | **a** | **b** | **c** | **c** | **b** | **c** | | --- | --- | --- | --- | --- | --- | --- | | **a** | ✅ | ❌ |  |  |  |  | | **b** |  | ✅ | ❌ |  |  |  | | **c** |  |  | ✅ | ✅ |  |  | | **c** |  |  |  | ✅ | ❌ |  | | **b** |  |  |  |  | ✅ | ❌ | | **c** |  |  |  |  |  | ✅ |   **Updated longest palindrome: len = 2** ("cc" at dp[2][3]). ****Step 3: Substrings of Length 3+ (****g ≥ 2****)**** For substrings of length g + 1, we check:  dp[i][j]=(str[i]==str[j]) AND dp[i+1][j−1]  **For g = 2 (substrings of length 3):**   |  | **a** | **b** | **c** | **c** | **b** | **c** | | --- | --- | --- | --- | --- | --- | --- | | **a** | ✅ | ❌ | ❌ |  |  |  | | **b** |  | ✅ | ❌ | ❌ | ✅ |  | | **c** |  |  | ✅ | ✅ | ❌ | ❌ | | **c** |  |  |  | ✅ | ❌ | ❌ | | **b** |  |  |  |  | ✅ | ❌ | | **c** |  |  |  |  |  | ✅ |   **Updated longest palindrome: len = 3** ("bccb" at dp[1][4]).  **For g = 3 (substrings of length 4):**   |  | **a** | **b** | **c** | **c** | **b** | **c** | | --- | --- | --- | --- | --- | --- | --- | | **a** | ✅ | ❌ | ❌ | ❌ |  |  | | **b** |  | ✅ | ❌ | ❌ | ✅ | ❌ | | **c** |  |  | ✅ | ✅ | ❌ | ❌ | | **c** |  |  |  | ✅ | ❌ | ❌ | | **b** |  |  |  |  | ✅ | ❌ | | **c** |  |  |  |  |  | ✅ |   **Updated longest palindrome: len = 4** ("bccb" at dp[1][4]).  **For g = 4 (substrings of length 5):**   |  | **a** | **b** | **c** | **c** | **b** | **c** | | --- | --- | --- | --- | --- | --- | --- | | **a** | ✅ | ❌ | ❌ | ❌ | ❌ |  | | **b** |  | ✅ | ❌ | ❌ | ✅ | ❌ | | **c** |  |  | ✅ | ✅ | ❌ | ❌ | | **c** |  |  |  | ✅ | ❌ | ❌ | | **b** |  |  |  |  | ✅ | ❌ | | **c** |  |  |  |  |  | ✅ |   **No update to len (remains 4).**  **For g = 5 (full string, length 6):**   |  | **a** | **b** | **c** | **c** | **b** | **c** | | --- | --- | --- | --- | --- | --- | --- | | **a** | ✅ | ❌ | ❌ | ❌ | ❌ | ❌ | | **b** |  | ✅ | ❌ | ❌ | ✅ | ❌ | | **c** |  |  | ✅ | ✅ | ❌ | ❌ | | **c** |  |  |  | ✅ | ❌ | ❌ | | **b** |  |  |  |  | ✅ | ❌ | | **c** |  |  |  |  |  | ✅ |   **Final longest palindrome: len = 4** ("bccb"). ****Final Answer**** The longest palindromic substring in "abccbc" has length 4 ("bccb").  Output: 4 |
| Output:- 4 | |