Check If a String is Palindrome

This C++ program checks if a given string is a palindrome using a recursive approach. A palindrome is a sequence of characters that reads the same forwards as it does backward.

Recursive function to check if a string is a palindrome

1. The program starts by including the necessary header files for input/output and string handling.
2. The **checkPalindrome** function is defined, which takes three parameters: **str** (the input string to be checked), **start** (the starting index for the current substring), and **end** (the ending index for the current substring).
3. Inside the **checkPalindrome** function, there are two base cases and one recursive case:
    - Base Case 1: If **start** is greater than or equal to **end**, it means the current substring has either one character (odd-length string) or no characters (even-length string). In both cases, it is considered a palindrome, so the function returns **true**.
    - Base Case 2: If the characters at the positions **start** and **end** are the same, the function makes a recursive call to **checkPalindrome** with **start** incremented by 1 (moving to the next character from the start) and **end** decremented by 1 (moving to the next character from the end).
    - Recursive Case: If the characters at **start** and **end** positions are not the same, the function returns **false**, indicating that the string is not a palindrome.
4. In the **main** function, two example strings **str1** and **str2** are defined.
5. The program then calls the **checkPalindrome** function for both **str1** and **str2**, starting with **start** as 0 and **end** as **str.length() - 1**.
6. Finally, the program prints whether each string is a palindrome or not based on the return value of the **checkPalindrome** function.

Time Complexity:

The time complexity of this palindrome-checking algorithm is O(n), where n is the length of the input string. The reason for this is that in the worst case, the recursive function will make a call for each character in the string, comparing characters from the beginning and the end, until the base cases are reached.

Space Complexity:

The space complexity of this algorithm is O(n) as well. This is because each recursive call adds a new frame to the call stack, and in the worst case, there will be n recursive calls,

each storing information about the characters' positions. Therefore, the space required is proportional to the length of the input string.

Recursive call stack for the approach

① Call stack for Recursive approach to check Palindrome

str = «aabbaa»

checkPalindrome («aabbaa», 0, 5)

|_ checkPalindrome («aabbaa», 1, 4)

|_ checkPalindrome («aabbaa», 2, 3)

|_ checkPalindrome («aabbaa», 3, 2)

|_ return true