Statement

↳ Given a string, s, return TRUE if it is a palindrome; otherwise, return FALSE.

Naive Approach

↳ Naive palindrome check involves:

- Removing non-alphanumeric characters
- Converting to lowercase for case-insensitive comparison.
- Reversing the cleaned string and comparing it to the original cleaned string.

↳ This approach requires:

- One full pass to build the cleaned string.
- Another pass to reverse it.
- O(n) time and O(n) space complexity.

↳ Downsides :

- Uses extra space for both the cleaned and reverse strings
- Performs redundant reversal operation.

↳ Less efficient than the two pointer method, which avoids reversal and reduces space usage.

Optimized Approach

↳ Two pointer approach optimizes both time and space usage

↳ Initialize two pointers:

- left at the start of the string
- right at the end of the string

↳ Move inward from both ends simultaneously

↳ Skip non-alphanumeric characters (e.g., spaces, punctuation)

↳ Convert characters to lowercase for case-insensitive comparison.

↳ Compare characters at left and right pointers:

- If they match, continue inward
- If they mismatch, return false immediately

↳ Stop when pointers meet or cross:

- If no mismatches were found, return true

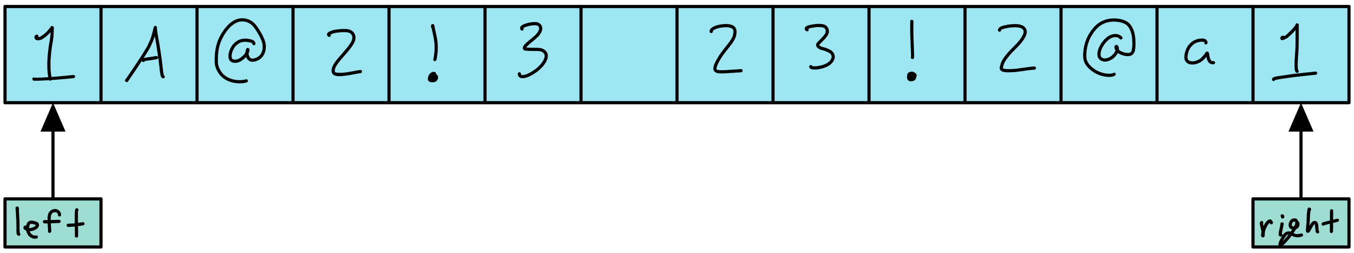
↳ Achieves O(n) time complexity and O(1) space complexity

↳ Highly efficient for checking palindromes in formatted strings

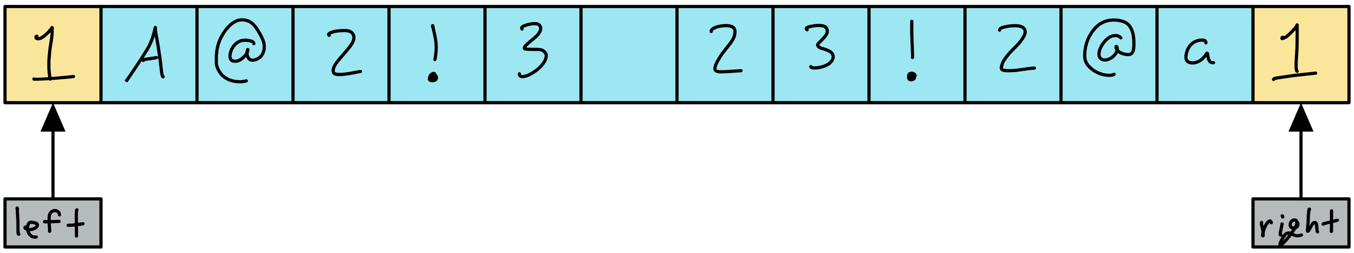
Visualization

↳ Check whether the string "1A@2!3 23!2@a1` is a palindrome

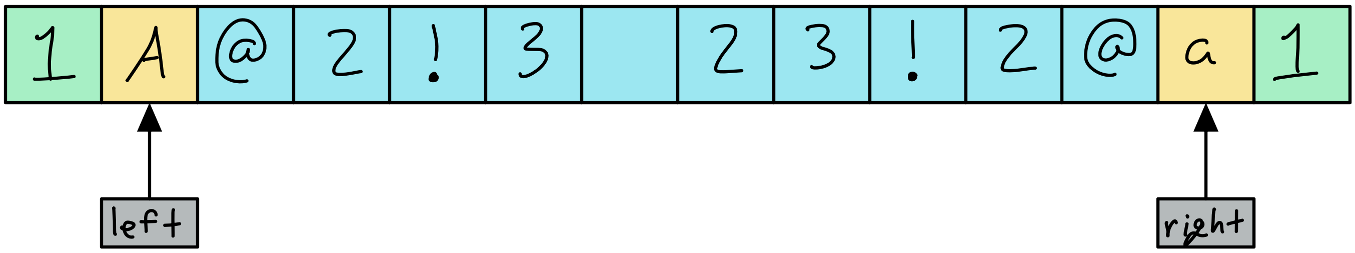
i) Start with two pointers: left at the beginning and right at the end. Compare the characters at both ends while ignoring non-alphanumeric characters.



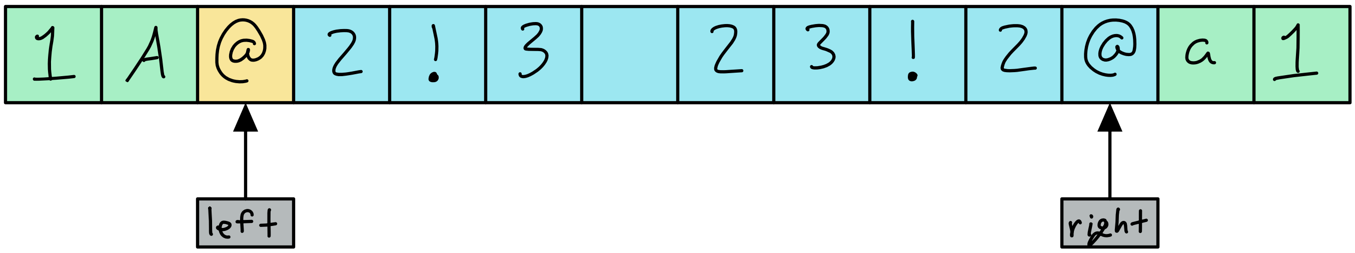
ii) Compare ´1` (left) and ´1` (right), both are valid and match. Move left pointer one step forward and right pointer one step backward.



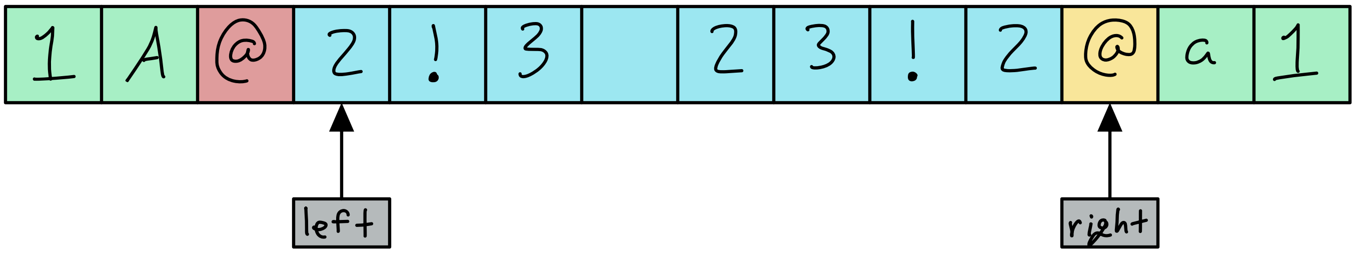
iii) Compare 'A` (left) and 'a` (right), both match ignoring case. Move left pointer one step forward and right pointer one step backward.



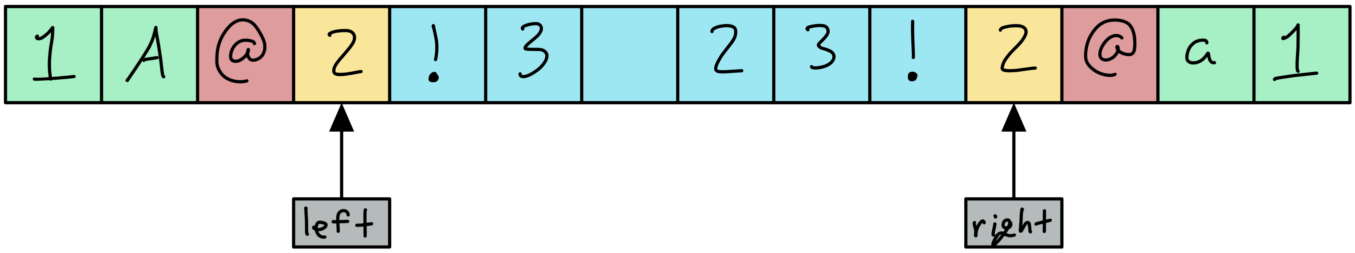
iv) Skip '@` (left), since it's not alphanumeric. Move left pointer one step forward.



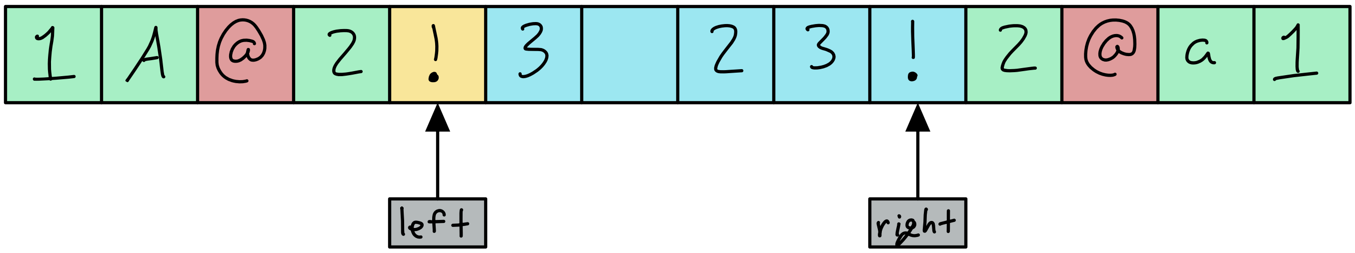
v) Skip '@` (right), since its not alphanumeric. Move right pointer one step backward.



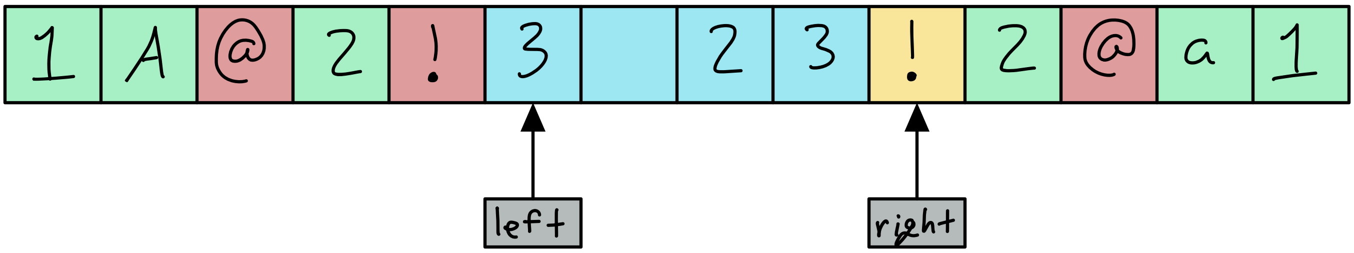
vi) Compare '2` (left) and '2` (right), they match. Move left pointer one step forward and right pointer one step backward.



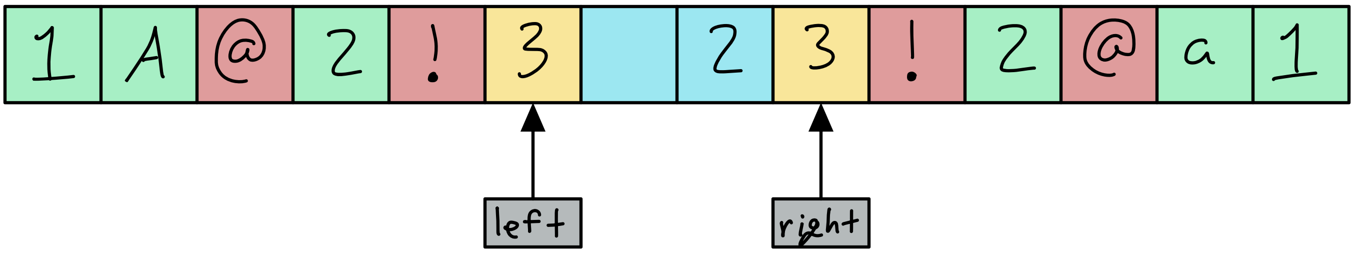
vii) Skip '!` (left), as it's not alphanumeric. Move left one step forward.



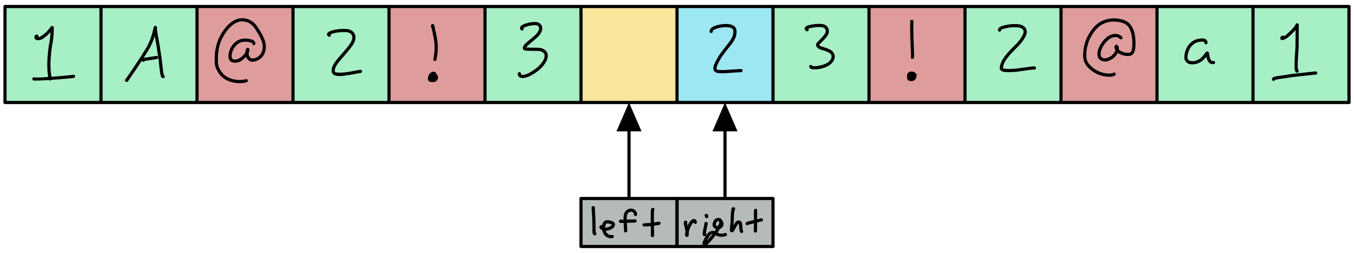
viii) Skip '!`(right), as it's not alphanumeric. Move right pointer one step backward.



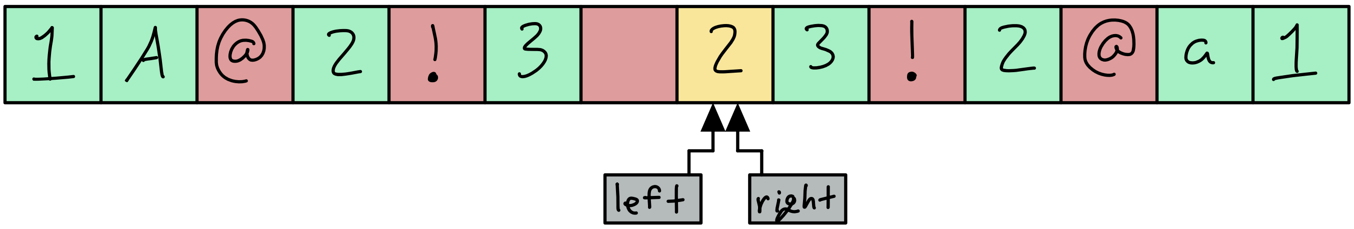
ix) Compare '3`(left) and '3`(right), they match. Move left pointer one step forward and right pointer one step backward.



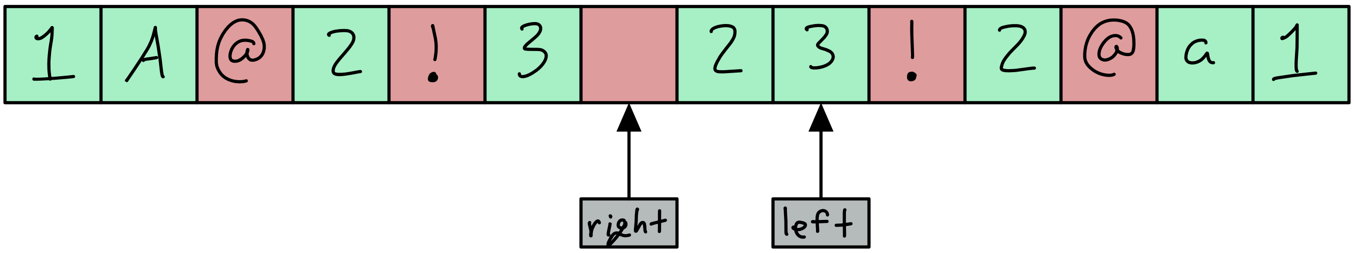
x) Skip space (left), as it's not alphanumeric. Move left pointer one step forward.



xi) Compare '2`(left) and '2`(right), they match. Move left pointer one step forward and right pointer one step backward.



xii) Now, the pointers have crossed each other. Since all valid characters matched successfully, the string is a palindrome.



Step-By-Step Solution

↳ Step 1: Initialize pointers and skip non-alphanumeric characters

i) Set up pointers:

- left starts at index Ø (beginning of the string)

- right starts at index len(s)-1 (end of the string)

ii) Process characters:

- If s[left] is not a letter or digit (e.g., space, punctuation, or special character), move left one step forward (left += 1). Repeat until left points to a valid character.

- If s[right] is not a letter or digit (e.g., space, punctuation, or special character), move right one step backward (right -= 1). Repeat until right points to a valid character.

↳ Step 2: Compare characters and move pointers

i) Convert both characters to lowercase, so the comparison is case-insensitive.

ii) Compare s[left] and s[right]:

- If they match, move both pointers inwards (left+=1, right-=1) to check the next pair.

- Return FALSE if they don't match, indicating the string is not a palindrome.

iii) This step repeats until the two pointers meet or cross each other. If all characters match, the function returns TRUE, confirming that the string is a palindrome.

Code (C++)

```
bool IsPalindrome(string s) {
    int left = 0, right = s.length()-1;

    while (left<right) {
        while (left<right && !alnum(s[left])) {
            left++;
        }
        while (left<right && !alnum(s[right])) {
            right--;
        }
        if (tolower(s[left]) != tolower(s[right])) {
            return false;
        }
        left++;
        right--;
    }
    return true;
}
```

Time Complexity

↳ The time complexity of the above solution is $O(n)$, where n is the number of characters in the string.

Space Complexity

↳ The space complexity of the above solution is $O(1)$.