*“Hard work beats talent when talent doesn't work hard”*

* *Tim Notke*

Dear reader, welcome to our first problem based on Strings. The problem name is *‘Print All Palindromic Substrings*’.

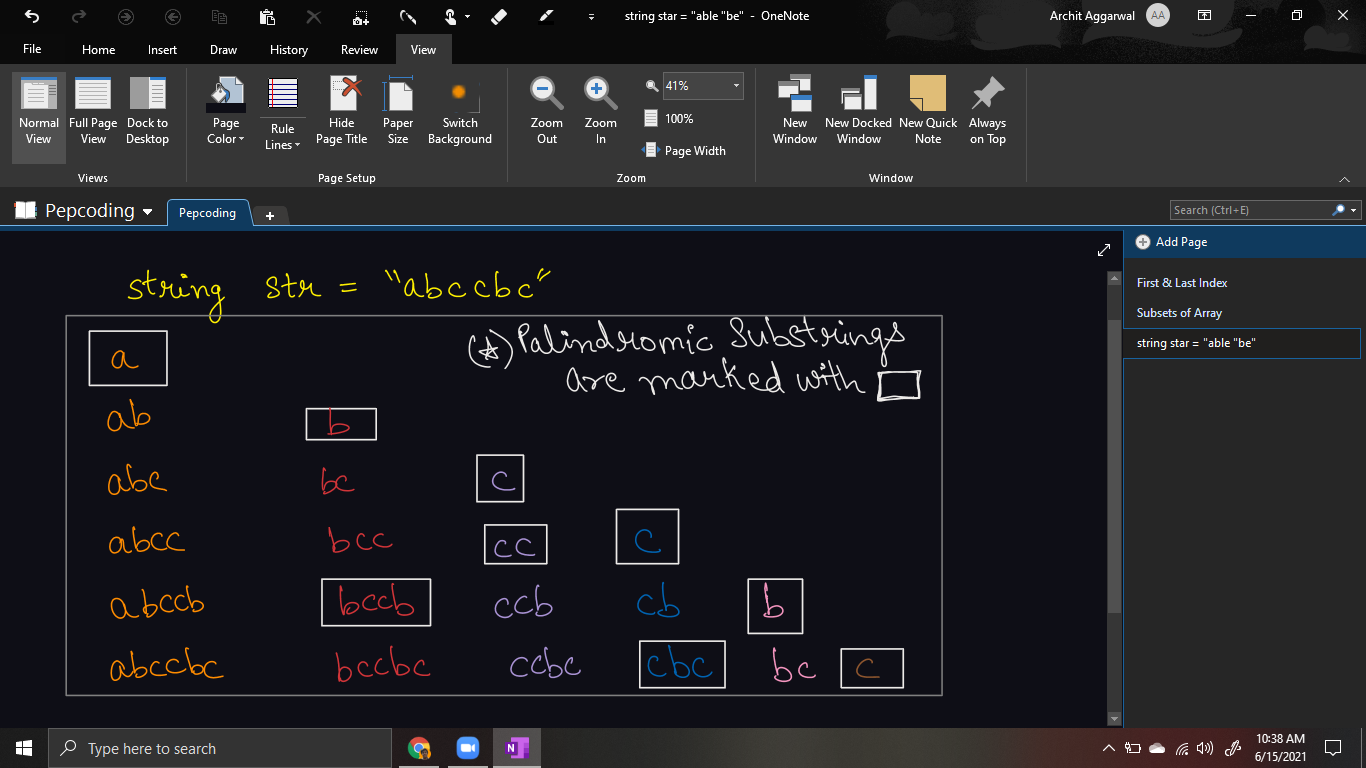
***Problem Statement***

* Given a string, print all palindromic substrings of it.
* A substring of a string is a contiguous subsequence of that string, i.e. it can be formed by deleting 0 or more characters from start and end of the string.
* For a string str = “abbc”, a substring can be “a”, “ab”, “abb”, etc. but “abc” cannot be a substring, as it is not contiguous. Also, “ba” is not a substring, as the order of characters is changed.
* A string is said to be a palindrome if the string read from left to right is equal to the string read from right to left. For example), strings “abbc”, “a”, “aba”, etc are palindromes, but strings “ab”, “abc”, etc. are not palindromes.

Important Links: [Problem Link](https://www.pepcoding.com/resources/online-java-foundation/string,-string-builder-and-arraylist/print-all-palindromic-substrings-official/ojquestion), [Solution Video](https://www.youtube.com/watch?v=_jM3iMT4k7g)

***Deducing an O(N3) Algorithm***

Let us take an example, consider a string s = “*abccbc*”. Try to write down all it’s substrings and find out the ones which are palindrome in nature.



Firstly, how to ***generate all the substrings***?

* As you can see, the substrings in the first column start with character ‘a’ at index 0, substrings in the second column start with character ‘b’ at index 1, and so on. Thus, the column can represent the ***starting index*** (= i).
* And within one column, the ending index (= j) of substrings in the rows, go from the starting index to the last character in the string, i.e. first 1 length substring starting with i, then 2 length substring, and so on. Hence a row can represent the ***ending index*** (or the length of substring).
* So, we can run two loops on string str, one for i from 0 to str.length - 1, and inner loop j from i + 1 to str.length, and extract the substring of str starting from i and ending before j.
* ***How*** to extract the substring from i to j-1? You might remember from the introductory video that there is an in-built function provided for doing just the same, and that is ***str.substring(i, j)***.
* ***Why*** are we starting the inner loop of j from i+1 ? Well because, the ending character’s index has to be greater than or equal to the starting character’s index. Otherwise, it will violate the condition of being a substring. (Remember! - “ba” is not a substring of “abbc”).

Now that you have generated all the substrings, the only task remaining is to check if the given substring is palindrome or not. How to ***check if a string is palindrome*** or not?

* We can maintain two pointers i and j which will start from 0 and string’s size - 1, and keep on incrementing i and decrementing j until they cross each other.
* We will keep checking whether characters at index i and j are equal or not. At any point, if we find they are not equal (strings do not read the same from left and right), we will return false.
* If pointers cross each other, it means that all the characters from left & right were equal. Hence return true.

Please try to code this without taking help of the video solution. It will help you develop an insight about substrings, palindromes, and nested loops.

***Implementation (Java)***

import java.io.\*;

import java.util.\*;

public class Main {

public static void solution(String str){

for(int i = 0 ; i < str.length(); i++){

for(int j = i + 1; j <= str.length(); j++){

// i -> starting point of substring, j-> ending point of substring

if(isPalindrome(str.substring(i, j))){

System.out.println(str.substring(i,j));

}

}

}

}

public static boolean isPalindrome(String str){

int i = 0, j = str.length() - 1;

while(i < j){

if(str.charAt(i) != str.charAt(j)){

return false;

}

i++;

j--;

}

return true;

}

public static void main(String[] args) {

Scanner scn = new Scanner(System.in);

String str = scn.next();

solution(str);

}

}

This code is written and explained by our team in [this video](https://www.youtube.com/watch?v=_jM3iMT4k7g) from *[3:05, 6:10]*. Please refer to it if you are stuck somewhere.

***Time & Space Complexity Analysis***

**Time Complexity** - Why O(n^3) Time?

We are running the outer loop from 0 to n-1 which takes O(n) and the inner loop from i to n-1, which will again take O(n).

Now, we are generating a substring from i to j, and checking whether it is a palindrome also takes O(j-i) = O(n) time.

Hence, the overall time complexity turns out to be O(n \* n \* n) = O(n^3) time.

**Space Complexity** - O(n) auxiliary space is required as we are passing the substring from i to j to isPalindrome function.

**Extra Gyaan(Knowledge)**: You should know that this solution is pretty bad in terms of time complexity. (Still, it is not worth starting with your journey of strings.) We have way better algorithms to do it in less complexity, but you will have to wait for them until the *Dynamic Programming* Section in Level 2 and *Manacher’s algorithm* in Level 3.

**Asked in Companies**: *Linkedin, Facebook*

I hope you enjoyed solving the problem with me. Dear reader, we will see you in the next problem: *String Compression*, until then keep coding!

Contributor : Archit Aggarwal