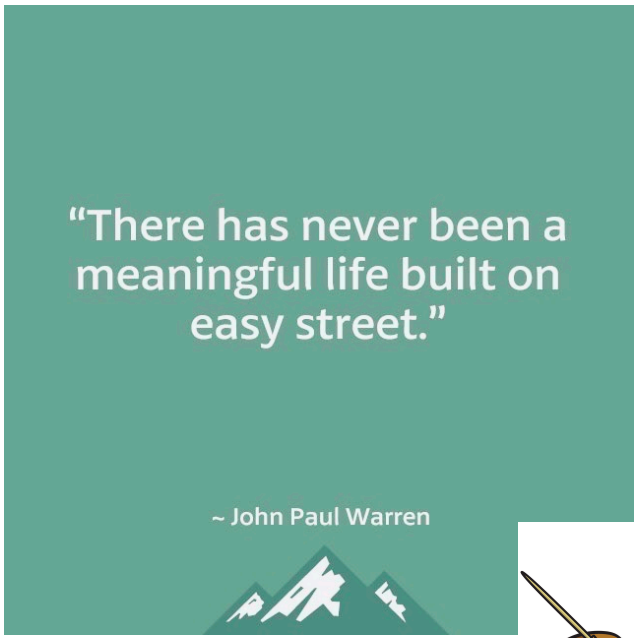
STRINGS



To do list

- → Introduction
- → Switch
- → Substring
- → check if string is palindrome
- → Longest Palindromic substring
- → Immutability of Strings

String → Array of characters

→ Sequence of characters

→ Collection / Group of characters ✗

"abcd" "bcad"

String str = "Welcome to Scaler"

Characters ⇒ Single symbol that represent a letter, a digit or a special letter

@ / ~

ch ⇒ 'A'

char ch2 = 'b'

char ch3 = '$'

Each & every character has some decimal associated with it. → ASCII Table

'A' → 65	'a' → 97	'0' → 48
'B' → 66	'b' → 98	'1' → 49
'C' → 67	'c' → 99	'2' → 50
⋮	⋮	⋮
'Z' → 90	'z' → 122	'9' → 57

'10' → combination of chars → string → enclosed in double " ".

* Concatenation

01. "a" + 1 ⇒ "a1"

02. "a" + "b" ⇒ "ab"

03. "a" + 'b' ⇒ "ab"

04. "a" + 2 + 3

"a2" + 3

"a23"

05 2 + 3 + "a"

5 + "a" ⇒ "5a"

06. char ch = (char) 65

print (ch) → A

07 char ch = (char) ('a' + 1)

(char) (98)

print (ch) → 'b'

08. int x = 'a'

print (x) ⇒ 97

* Switch Case

Given a char[ ], toggle every character of array

↓

uppercase to lowercase

lowercase to uppercase

Note :- all characters are either in uppercase or lowercase.

char[ ] = " sCAlEr "

↓

"SCaleR"

[aDgbHJe] → [AdGBhjE]

* Idea →

Check if ch is in uppercase

// ch ≥ 65 && ch ≤ 90

// ch ≥ 'A' && ch ≤ 'Z'

Convert it to lowercase

ch + 32

ch − 65 + 97

ch - 'A' + 'a'

0 + 97

01. ch = 'A' ⟶ 'A' − 'A' + 'a'

0 + 'a' = 'a'

02. ch = 'B' ⟶ 'B' − 'A' + 'a'

1 + 97 = 98 → 'b'

Lowercase → {TODO}

```
void toggle (char[] str){
    for (i=0; i<n; i++){
        ch = str[i];
        if (ch ≥ 'A' && ch ≤ 'Z'){
            str[i] = (char)(ch - 'A' + 'a');
        }
        else if (ch ≥ 'a' && ch ≤ 'z'){
            str[i] = (char)(ch - 'a' + 'A');
        }
    }
}
```

TC: O(N)

SC: O(1)

TODO → { Solve it using Bit Manipulation in O(1) TC }

* Substring → Continous part of a string within a string

↳ concepts will be same with subarray

→ Single char is also a substring

→ Entire string is also a substring

String str = "abc"

"a" "b" "c"

"ab" "bc"

"abc"

* No. of substring for "bxcd"

"b" "x" "c" "d"

"bx" "xc" "cd"

"bxc" "xcd"

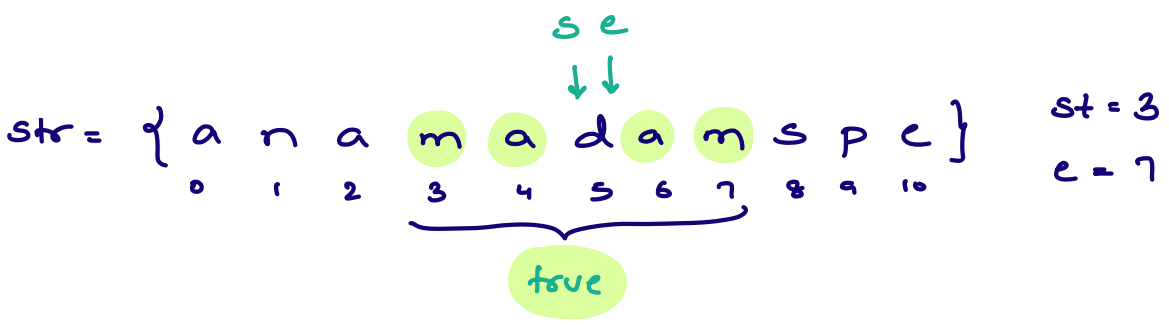
"bxcd"

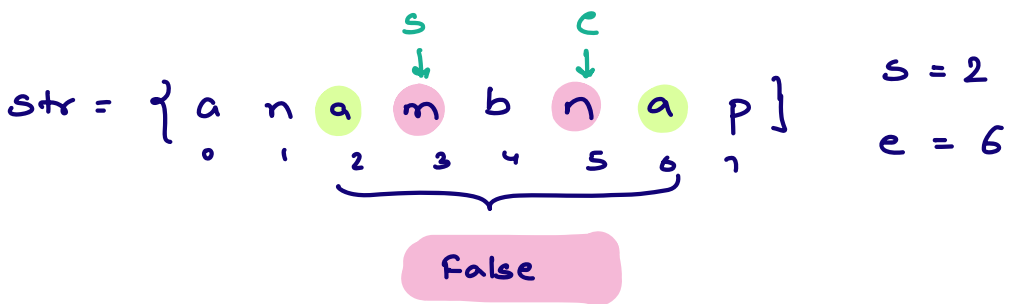
4 + 3 + 2 + 1 ⇒ 10

No. of substring = $\frac{n*(n+1)}{2}$

* Check whether the given substring is palindrome or not

palindromes = "level"
= "madam"
= "mom"
= jahaj
= racecar






```
boolean ispali ( String str, int s, int e )
    while ( s < e ) {
        char ch1 = str[s];
        char ch2 = str[e];
        if ( ch1 != ch2 ) return false
        s++;
        e = e-1;
    }
    return true;
}
```

TC: O(n)
SC: O(1)

8:10 → 8:20 AM

Q Given a string, calculate the length of **longest palindromic substring**

Constraints $1 \le N \le 3 * 10^3$

For String str
LPS = Entire string
SPS = 1

Eg:- a b a c a b d → 5
0 1 2 3 4 5 6

Eg:- a b c d → 1
0 1 2 3

Eg:- { f e a c a b a c a b g f } → 7

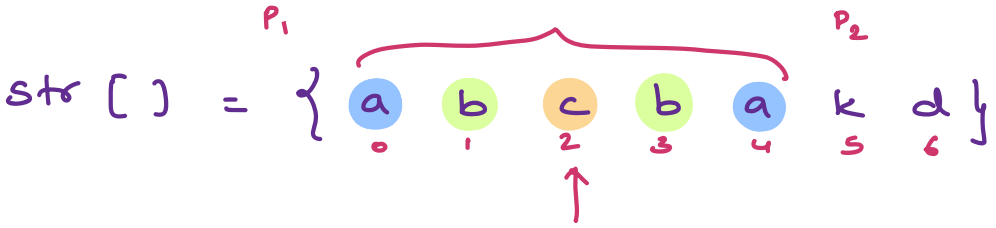
Eg:- { a d a e b c d f d c b e f g g f e } → 9

Naive Approach → Get all substrings & check if it is a palindrome or not.

```
int longestpali (String[] str){
    int ans=0
    for (i=0; i<n; i++){
        for (j=i; j<n; j++){   // sub [i j]
            if (ispali(str, i, j)==True){
                ans = Math.max(ans, j-i+1);
            }
        }
    }
    return ans;
}
```

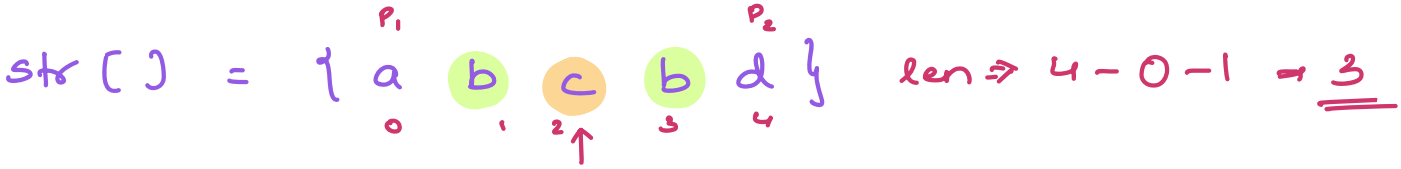
TC: $O(N^3)$
SC: $O(1)$
TLE

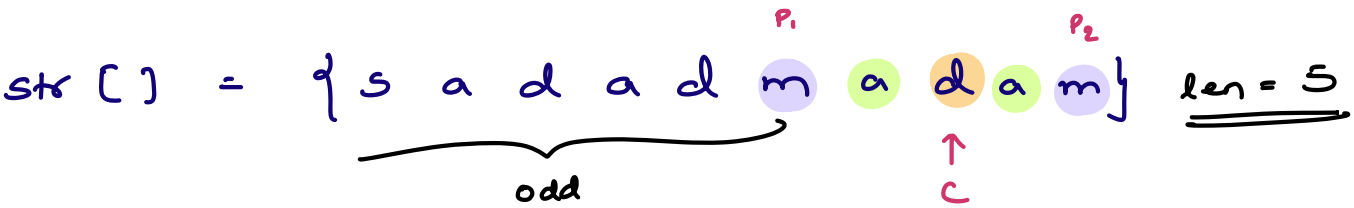
* Optimal Approach → Expand the palindromic from center

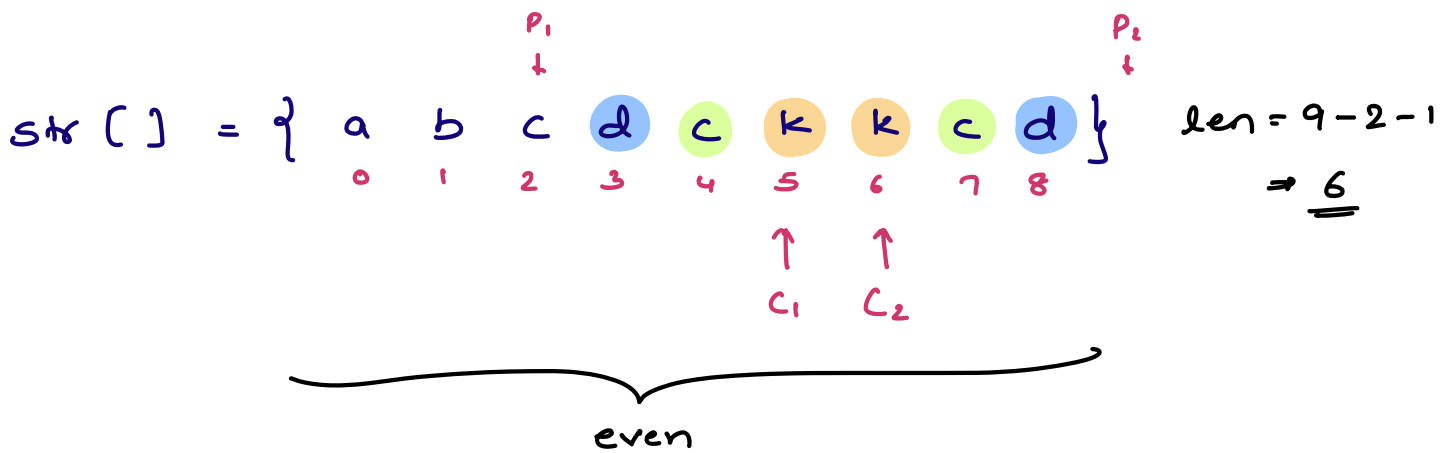


included $[P_1\ P_2]$ ⇒ len = $P_2 - P_1 + 1$

excluded $(P_1\ P_2)$ ⇒ len = $P_2 - P_1 + 1 - 2$ ⇒ $P_2 - P_1 - 1$







Obs ⇒ $(P_1\ P_2)$ = len = $P_2 - P_1 - 1$

* Palindrome ⇒ odd length → c_1

even length → c_1 & c_2

* **Odd length palindromes**

  ↳ Take every character as center & try to expand window around center & get max possible longest palindromic length

* **Even length palindromes**

  ↳ Take every two adjacent chars as the center & try to expand the window around c_1 & c_2. Get max possible palindromic length;

```
int longestpal ( String str)
{
    int ans = 0

    for ( i=0; i<n; i++) {          // odd len palindromes
        // center = i.
        P1 = i
        P2 = i
        len = Expand ( str, P1, P2 );
        ans = Math.max ( ans, len);
    }
```

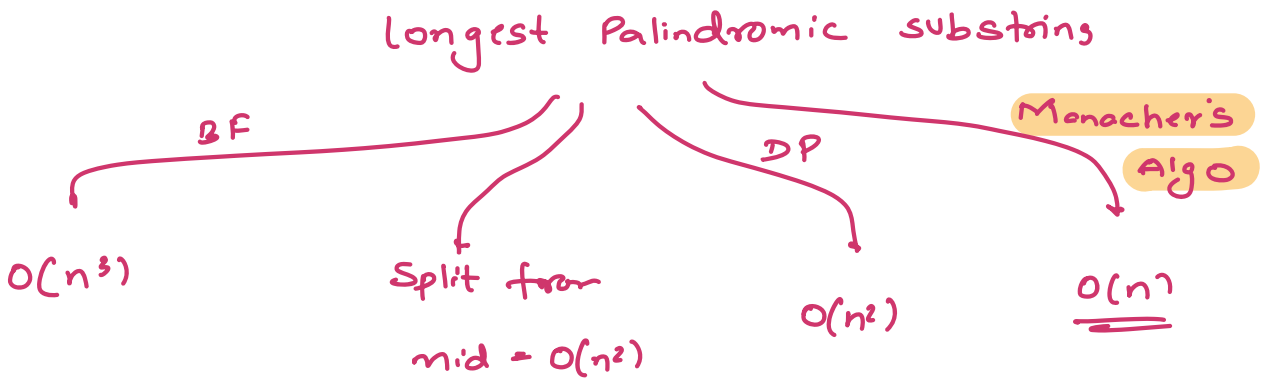
TC : $O(n^2)$

SC : $O(1)$

```
    for ( i=0 ; i<n-1 ; i++){      // even length
        // centers => i & i+1
        p1 = i
        p2 = i+1
        len= Expond ( str , p1 , p2 );
        ans= Math.max ( ans, len);
    }
    return ans;
}

int expand ( string str, p1 , p2 )
{
    while ( p1 >=0 && p2 <n && str[p1] == str[p2] )
    {
        p1 = p1 - 1;
        p2 = p2 + 1;
    }
    return p2 - p1 - 1;
}
```

TC: O(n)

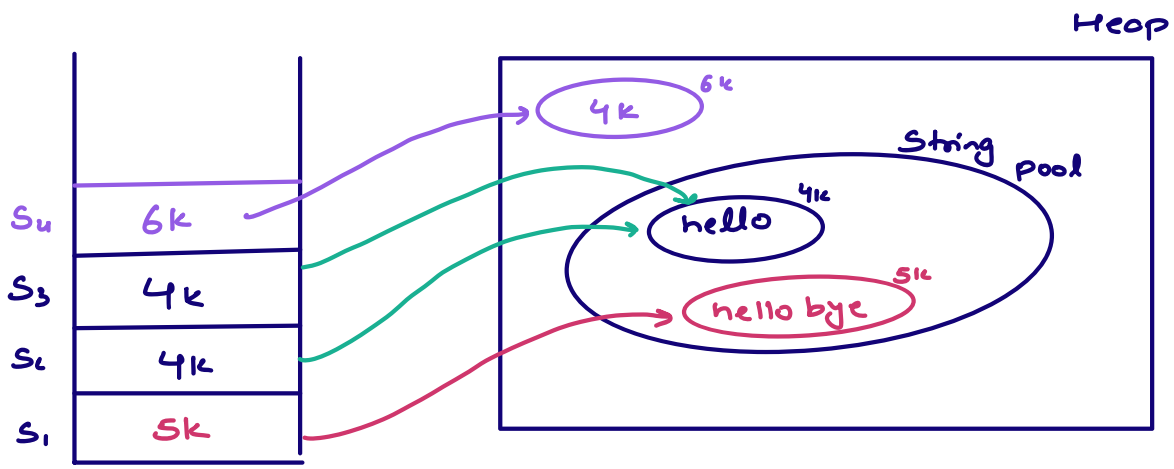


* Immutability of Strings

→ JS, Java, C#, Python & Go - Strings are **immutable**

Values can't be changed.

```
String s1 = "hello";
String s2 = "hello"
String s3 = s1
String s4 = new String("hello");
```



s1 = s1 + "bye";

* To manage the memory inside string pool,

→ If a string is already present in string pool, its memory will be shared

* Why Immutability → Many references are pointing to the same string, so changing

the string through any reference will leads to shock for the other references.

print (S_2 == S_3) → True

print (S_2 == S_4) → False

} Not correct way of comparison

print (S_2 . equals (S_3)) → True

print (S_2 . equals (S_4)) → True

* Doubts * —— * —— *

$S_5 = S_4$

$S_5 = 612$