SCALER

# String Basics

TABLE OF CONTENTS

1. Strings
2. Questions on Strings

# String

A string can be defined as a sequence of characters or in other words we can say that it is an array of characters.

C/C++

Example
Below are the examples of string:

"Welcome to Scaler"
"Hello World!"

Note: String is represented using double quote i.e, "". All the characters of string must be inside this quote.

# Character

A character is a single symbol that represents :

- Alphabets
  - Lowercase → 'a' 'b' 'c' .....
  - Uppercase → 'A' 'B' 'C' .....
- Digits [0 - 9] → '0' '1' '2' '3' .....
- Special Characters → '@' '#' '$' .....

SCALER

**Computer stores everything in Binary. How do we store strings in computer?**

# ASCII

Each character has corresponding decimal value associated to it which is known as ASCII value.

| | | |
|---|---|---|
| 'A' : 65 | 'a' : 97 | '0' : 48 |
| 'B' : 66 | 'b' : 98 | '1' : 49 |
| 'C' : 67 | 'c' : 99 | '2' : 50 |
| 'D' : 68 | 'd' : 100 | ⋮ |
| 'E' : 69 | 'e' : 101 | ⋮ |
| ⋮ | ⋮ | ⋮ |
| 'Y' : 89 | 'y' : 121 | '9' : 57 |
| 'Z' : 90 | 'z' : 122 | |

SCALER

1. char ch = 'a';

   print(ch);  o/p : 'a'

2. char ch = (char) 65;

   print(ch);  o/p : 'A'

3. int ch = 'a';  a - 97

   print(ch);

   o/p : 97

4. int ch = 'a' + 1;  = 98

   97

   print(ch);

   o/p : 98

INT + LONG → LONG

CHAR + INT → INT.

## Switch Case

**< Question > :** Given a string consisting of lower-case and upper-case alphabets.

Convert: (1) lowercase → uppercase

(2) uppercase → lowercase

1. **"H e l l o"** - hELLO

QUIZ

2. **"a D g b H J e"** - AdGBhjE

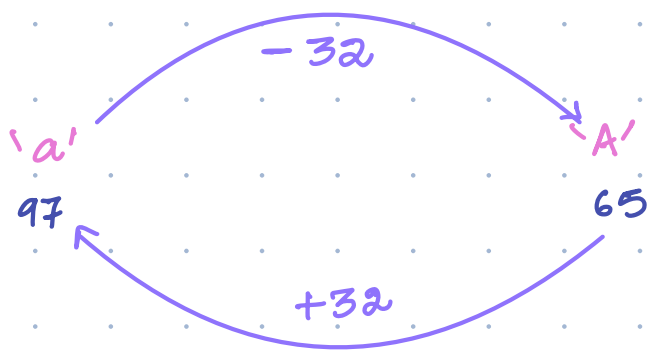
Observations:

Step 1: Identify the current case of characters. - ASCII values

65 - 90 → Uppercase

97 - 122 → Lowercase.

Step 2: Toggle the case.




L $\xrightarrow{-32}$ U

U $\xrightarrow{+32}$ L

SCALER

```
String Toggle(String s) {
    String ans = "";
    for (i = 0; i < s.length(); i++) {
        if (s[i] >= 'A' && s[i] <= 90) {
            ans += (char)(s[i] + 32);
        }
        else {
            ans += (char)(s[i] - 32);
        }
    }
    return ans;
}
```

NOTE : For C++ :

Above sol$^n$ works fine with T.C : O(N) & S.C : O(1)

For languages like Java, where strings are IMMUTABLE :

In above code, we are making changes to same given string. This is not possible in languages like Java, Python etc where strings are IMMUTABLE. Due to STRING IMMUTABILITY concept, above code will create a new string everytime a character is added to ans string, which means adding 1 char can take O(N) time.

$\therefore$ T.C = $O(N^2)$ S.C = $O(N^2)$

SCALER

## Optimised Solution:

Whenever we have to Modify String:

Convert String to char[] → Modify → Convert char[] to string

```
String Toggle(String s) {
    // Convert String to char[]
    char[] ch = s.toCharArray();          → O(N)
    for (i = 0 ; i < ch.length ; i++) {
        if (ch[i] >= 'A' && ch[i] ≤ 90) {
            ch[i] = ch[i] + 32;
        }
        else {
            ch[i] = ch[i] - 32;
        }
    }                                     O(N)
    // Convert Char[] back to String
    return new String(ch);                → O(N)
}
```

T.C: O(N+N+N) = O(3N) = O(N)

S.C: O(N) // for char[]

SCALER

# Substring

A substring is a continuous sequence of characters within a string.

Substring concept is similar to Subarray concept in array.

A single character is also a substring.

Whole string is also a substring.

NOTE : Empty string (" ") is not a substring.

```
        0 1 2
String str = "a b c"
```

Substrings of str:

[0 0] → a  [1 1] → b  [2 2] → c

[0 1] → ab  [1 2] → bc

[0 2] → abc

QUIZ:

```
        0 1 2 3
String str = "b x c d"
```

Substrings of str:

[0 0] → b  [1 1] → x  [2 2] → c  [3 3] → d

[0 1] → bx  [1 2] → xc  [2 3] → cd

[0 2] → bxc  [1 3] → xcd

[0 3] → bxcd

Total No of substrings for string of len = N : $\frac{N(N+1)}{2}$

SCALER

## Check whether given string is palindrome or not.

Sequence of chars that reads same forward & backward.

Eg: "madam"

"malayalam"

start end

Approach:

Case 1: s[start] == s[end] → start++ end--

Case 2: s[start] != s[end] → break

```
bool checkPalindrome(String str){
    int start=0, end = str.length()-1;

    while(start < end) {
        if (str[start] != str[end])
            return false;

        start++;
        end--;
    }
    return true;
}
```

T.C: O(N)

S.C: O(1)

SCALER

## Check whether the substring from index s to e of given string is palindrome or not.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| str = " | a | n | a | m | a | d | a | m | s | p | e " |

s = 3
e = 7

```
bool checkSubstrPalindrome( String str, int s, int e){
    int start = s, end = e;

    while(start < end) {

        if (str[start] != str[end])
            return false;

        start++;
        end--;
    }
    return true;
}
```

T.C: O(N)
S.C: O(1)

SCALER

**Given a centre.Find the max length of the palindrome centred at that position.**

Constraints: $0 \le i \le j \le N$

$j - i \le 1$

Eg:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | b | d | y | z | z | y | d | e | b | d | y | z | y | d | x |

$i = 4$
$j = 5$

ans : 6

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | b | d | y | z | z | y | d | e | b | d | y | z | y | d | x |

$i = 12$
$j = 12$

ans : 5

Approach: Starting from centre, keep on expanding in both directions till elements are same.

When elements are different, calculate length.

Length of string from $[i+1, j-1]$ : $j-1-(i+1)+1 = j-i-1$.

```
int getPalindromeLength(String s, int i, int j) {
    if(s.length() == 0)  return 0;      // Edge case

    while( s[i] == s[j]  && i >= 0  && j < s.length()) {
        i--;
        j++;
    }

    return j-i-1;
}
```

T.C : O(N)

S.C : O(1)

## Dry Run:

S = y a b b a x

0 1 2 3 4 5

i j

ans: 5-0-1 = 4

| i | j | S[i] == S[j] && i >= 0 && j <= N |
|---|---|---|
| 2 | 3 | ✓ |
| 1 | 4 | ✓ |
| 0 | 5 | ✗ |

S = y a b a x

0 1 2 3 4

i j

ans: 4-0-1 = 3

| i | j | S[i] == S[j] && i >= 0 && j <= N |
|---|---|---|
| 2 | 2 | ✓ |
| 1 | 3 | ✓ |
| 0 | 4 | ✗ |

S = y c b a x

0 1 2 3 4

i j

ans: 3-1-1 = 1

| i | j | S[i] == S[j] && i >= 0 && j <= N |
|---|---|---|
| 2 | 2 | ✓ |
| 1 | 3 | ✗ |

S = y a b c a x

0 1 2 3 4 5

i j

ans: 3-2-1 = 0

| i | j | S[i] == S[j] && i >= 0 && j <= N |
|---|---|---|
| 2 | 3 | ✗ |
| | | |
| | | |

SCALER

**< Question > :** Given a string s. Find the length of the longest palindrome substring in s.

$1 \le N \le 10^3$

```
        0 1 2 3 4 5 6 7 8
str = " a n a m a d a m m "      ans : 5
```

QUIZ:

```
        0 1 2 3 4 5 6 7 8 9 10 11
str = " f e a c a b a c a b g f "      ans : 7
```

QUIZ:

```
        0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
str = " a d a e b c d f d c b e t g g t e "      ans : 9
```

[ BF Idea ] - Consider all substrings of given string & check which one of them are Palindrome. Check the length of Palindromic substring to get Max length of Palindromic substring.

```
int LongestPalindrome( String str) {
    N = str.length();
    ans = 1;
    for( s = 0 ; s < N ; s++) {
        For( e = s ; e < N ; e++) {
            || Check Palindrome
            if ( checkSubstrPalindrome (Str, s, e)) {
                ans = Math.max( ans , e - s + 1) ;
            }
        }
    }
    return ans ;
}
```

T.C : $O(N^3)$
S.C : $O(1)$

SCALER

## Optimised Approach:

a b a e a b f

x b d y z z y d b d y z y d x

**Observations:**

b a e a b

centre

Odd Length Palindrome

b a e e a b

centre

Even Length Palindrome.

- Palindromic strings are symmetric about its centre.

Given a centre, we can get length of largest Palindrome centered at that position. Since we are not sure of centre of Palindromic substring in given string we consider all possible centres.

Every index can be possible centre.

**Approach:** Taking every index as CENTRE, check for LARGEST length of Palindrome centered at that position. For every centre, check both EVEN & ODD length Palindrome & consider max of both.

< / > Code

```
int LongestPalindrome( String str) {

    ans = 1;
    for ( i=0 ; i < str.length; i++) {

        OddLength = getPalindromeLength(str, i, i);

        EvenLength = getPalindromeLength(str, i, i+1);

        ans = Int.Max( ans, max(OddLength, EvenLength));
    }

    return ans;
}
```

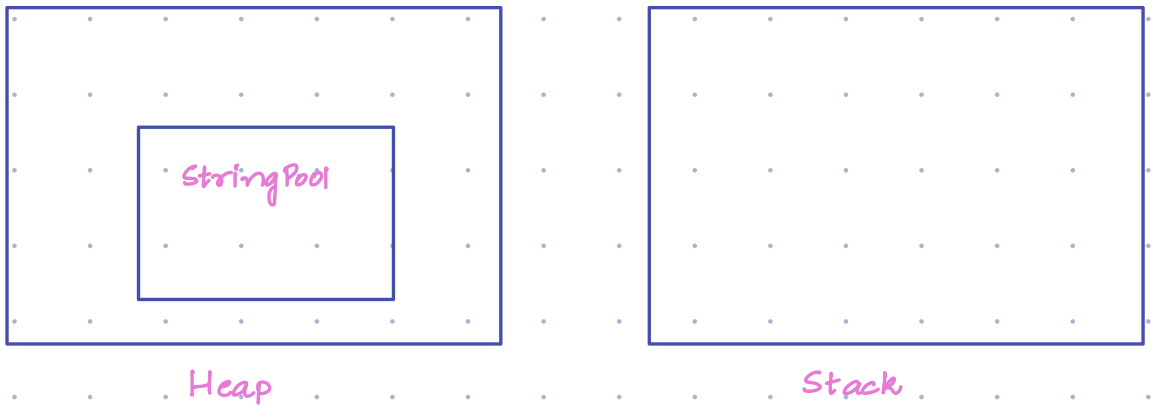
T.C : $O(N^2)$

S.C : $O(1)$

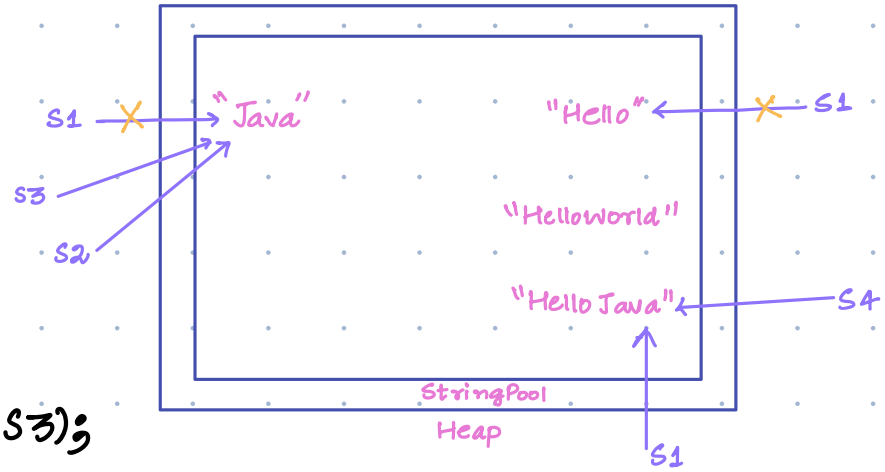
## Immutability of the strings

In languages like Java, C#, JavaScript, Python and Go, strings are immutable, which means that once a String object is created, its value cannot be changed.

If you try to modify it, such as through concatenation or other operations, a new String object is created with the modified value, and the original String remains unchanged.

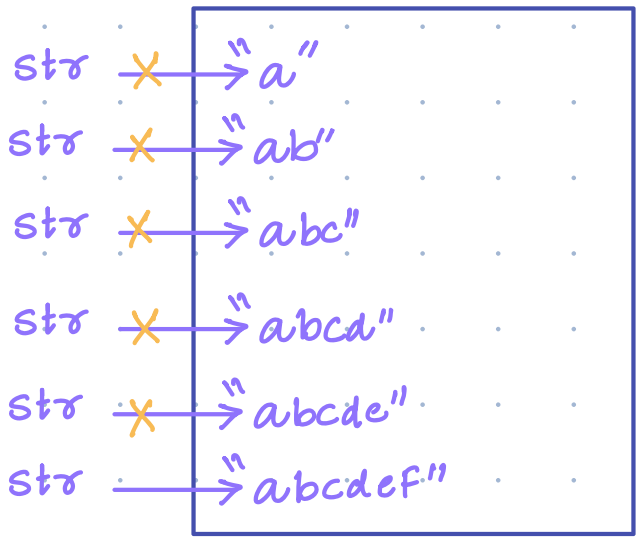



```
String s1 = "Java";
String s2 = "Java";
String s3 = s1;
      s1 = "Hello";
 s1.concat("World");
String s4 = s1.concat(s3);
      s1 = s4;
```



# QUIZ:

Considering the immutability of strings in most programming languages, if you start with an empty string and keep adding one character at a time, how many string objects will be created when forming the string "abcdef"?




ans: 6

Doubts

$a = 10$

| | |
|---|---|
| 2 | 10 |
| 2 | 5 — 0 |
| 2 | 2 — 1 |
| 2 | 1 — 0 |
| | 0 — 1 |

0 - - - - 00001010 | 1

0 - - - - - 00000001

0 - - - 00001011