Todays Content

1. Basic strings
2. longest palindromic substring
3. 1st repeating char

String in C++;

```
                0 1 2 3 4
string  s = "H e l l o";

print(s[1])  # e
print(s);  "Hello"
print(s.length()) ; 5
```

0 1 2

string s_1 = "a n t"

String s_2 = "a c t"

<, <=, >, >=, !=, ==

Note: String m Relational operator, it will compare char by char, their ascii

print(s_1 == s_2) #false

print(s_1 > s_2) # True

Dictnory cmp, lexicographical order. TC: O(N) lngh of string

String Concatenation:

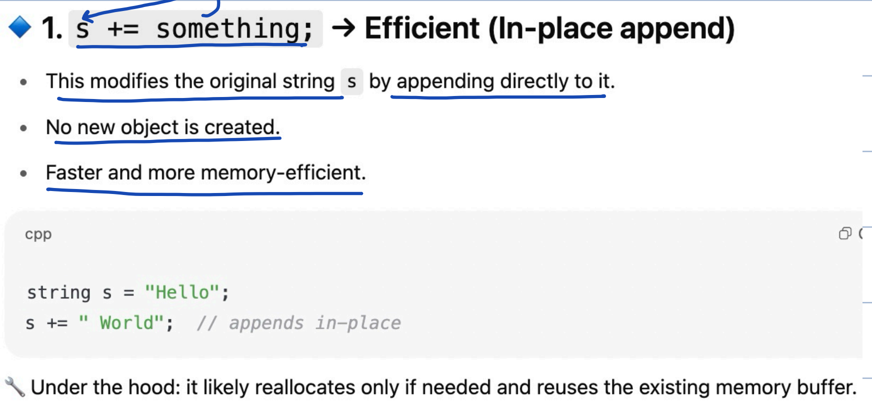
#Way1:

S = "Hello"   S> Hellow

S += "W"

#S = "Hello W"

TC: O(1)



🔷 1. `s += something;` → **Efficient (In-place append)**

- This modifies the original string `s` by appending directly to it.
- No new object is created.
- Faster and more memory-efficient.

```cpp
string s = "Hello";
s += " World";  // appends in-place
```

🔧 Under the hood: it likely reallocates only if needed and reuses the existing memory buffer.

#Way2:

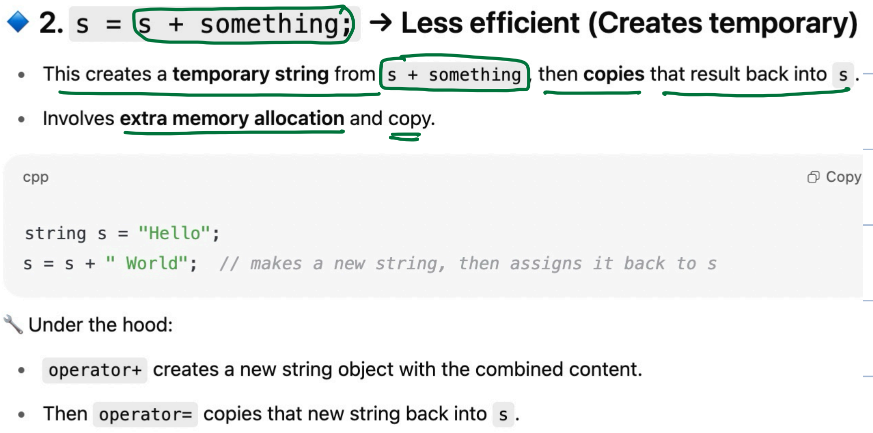
S = "Hello"   s = Hello   Hellow

S = S + "W"

#S = "Hello W"   S> Hellow

N is lngh of string befre concat

TC: O(N+1)



🔷 2. `s = s + something;` → **Less efficient (Creates temporary)**

- This creates a **temporary string** from `s + something`, then **copies** that result back into `s`.
- Involves **extra memory allocation** and copy.

```cpp
string s = "Hello";
s = s + " World";  // makes a new string, then assigns it back to s
```

🔧 Under the hood:

- `operator+` creates a new string object with the combined content.
- Then `operator=` copies that new string back into `s`.

Q. Why Cpp implicit type casting not there

Cpp vs Java

18 Given a String, return length of longest Palindromic substrings

Subarray concept in Strings

Constraints:

$1 <= N <= 10^3$

```
                 0 1 2 3 4 5                       0 1 2 3
Ex1:        s =  a b a c a b    ans = 5    Ex2 s = a n b c      ans = 1

                  0 1 2 3 4
Ex3         s =   d a b b a     ans = 4            a a b b
```

Idea: Generate all substrings

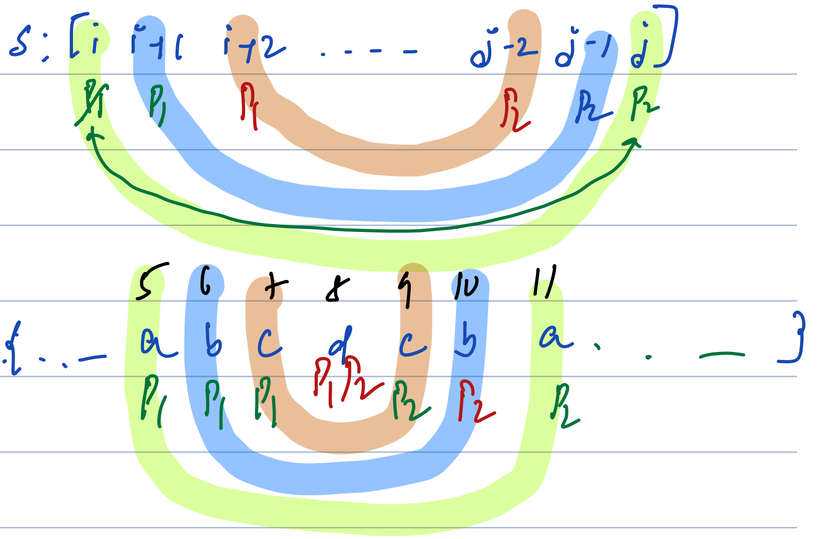
For each string check palindrome or not

TC: $O(N^2 * N) = O(N^3)$ SC: $O(1)$

N^2 → # No: of substrings

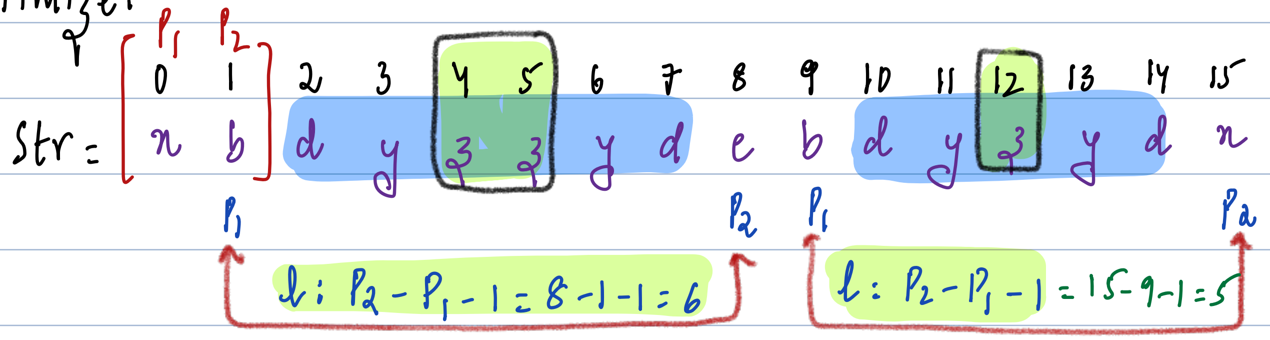
N → # Time taken to check if substring palindrome or not

```
int longPal(string s){
    int ans = 0, N = s.length();
    for(int i = 0; i < N; i++){
        for(int j = i; j < N; j++){
            # sub[i - j] check palindrome or not:
            int p1 = i, p2 = j;
            bool ispal = true;
            while( p1 < p2){
                if(s[p1] == s[p2]){
                } p1++; p2--;
                else{
                    ispal = false;
                } break;
            }
        } if(ispal == true){ ans = max(ans, j-i+1)}  # substring [i-j] is pal
    }
} return ans;
```



s: [i i+1 i+2 . . . . j-2 j-1 j]
P1 P1 P1 P2 P2 P2
5 6 7 8 9 10 11
s : { . . _ a b c d c b a . . _ }
P1 P1 P1 P1P2 P2 P2 P2

Optimize:



$$\begin{array}{c|cccccccccccccccc} & P_1 & P_2 & & & & & & & & & & & & & & \\ & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ Str = & x & b & d & y & z & z & y & d & e & b & d & y & z & y & d & x \\ & & P_1 & & & & & & & P_2 & P_1 & & & & & & P_2 \end{array}$$

$l: P_2 - P_1 - 1 = 8-1-1 = 6$

$l: P_2 - P_1 - 1 = 15-9-1 = 5$

Idea TC: $O(N*N + N*N) = O(N^2)$ SC: $O(1)$

1. Maximum odd length palindromic substring.
   Take every char as center:
   Expand on centres & calulate max palindrome length
   & get overall max for all centres.

2. Maximum even length palindromic substring.
   Take every adjacent pair as centre
   Expand on centres & calulate max palindrome length
   & get overall max for all centres.

3. max(max odd length palindrome, max even length palindrome)

Expand on Center?

$$\begin{array}{c|ccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ S = & a & b & e & f & g & f & e & k & l \end{array}$$

$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ S = & a & b & c & b & a \end{array}$$

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ S = & a & b & c & d \end{array}$$

```
int longPal(string s){

}
```

2B Given a String s:

Return 1st repeating character.

String s_1 =

String s_2 =

String s_3 =