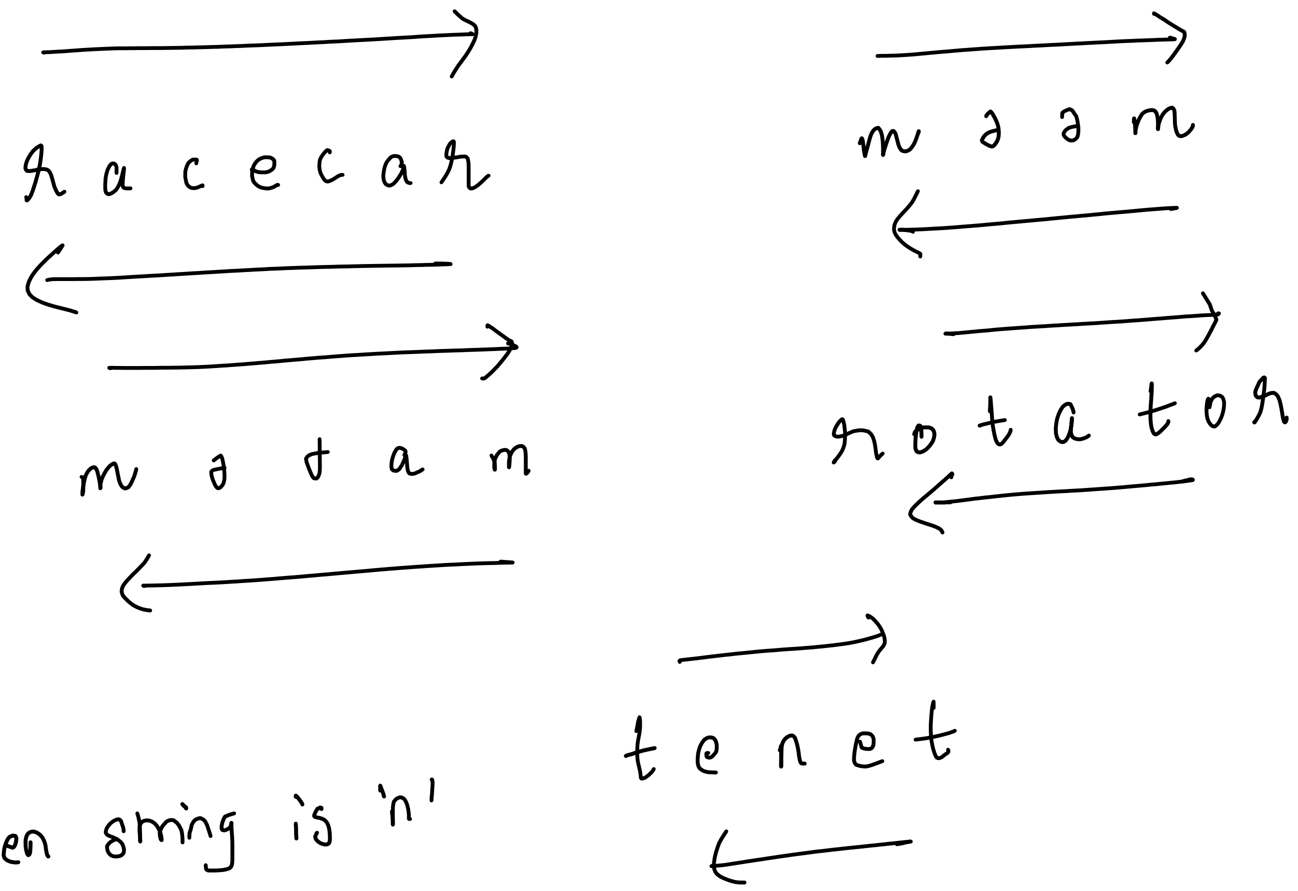
Check Palindrome

⇒ Given a string represented as a character arrays, check it is a **palindrome** or not.

Example

Input : "racecar"
Output : true

Input : "rotator"
Output : true



↳ assume len of the given string is 'n'

1. create a copy of the given string ⇒ n
2. reverse the copy ⇒ n/2
3. compare the reversed copy with the given string ⇒ n

$2n + n/2$ steps

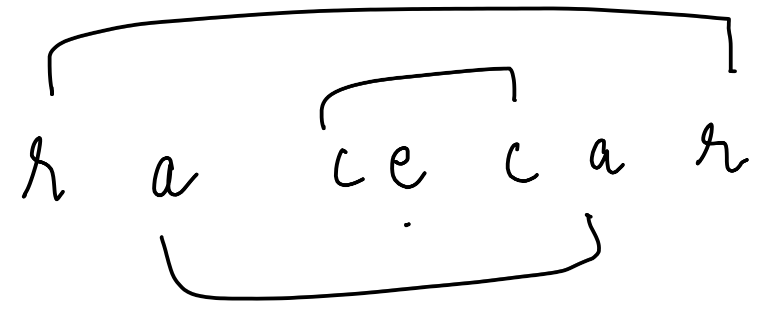
~ O(n)

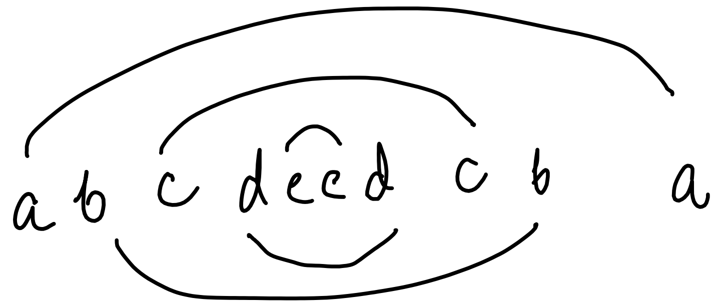
space : O(n)

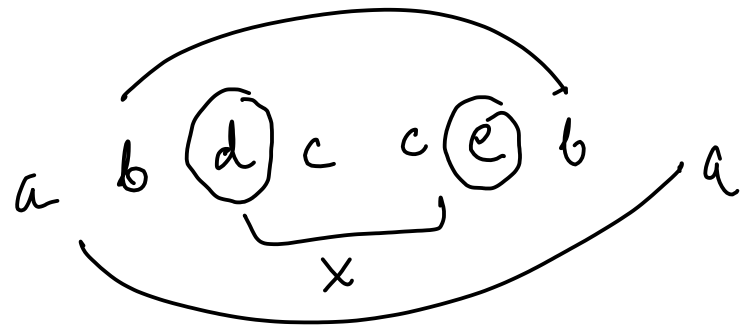
match → true (palindrome)

no match → false (not a palindrome)

2nd







$i = 0$ $j = n-1$ → len. of the string

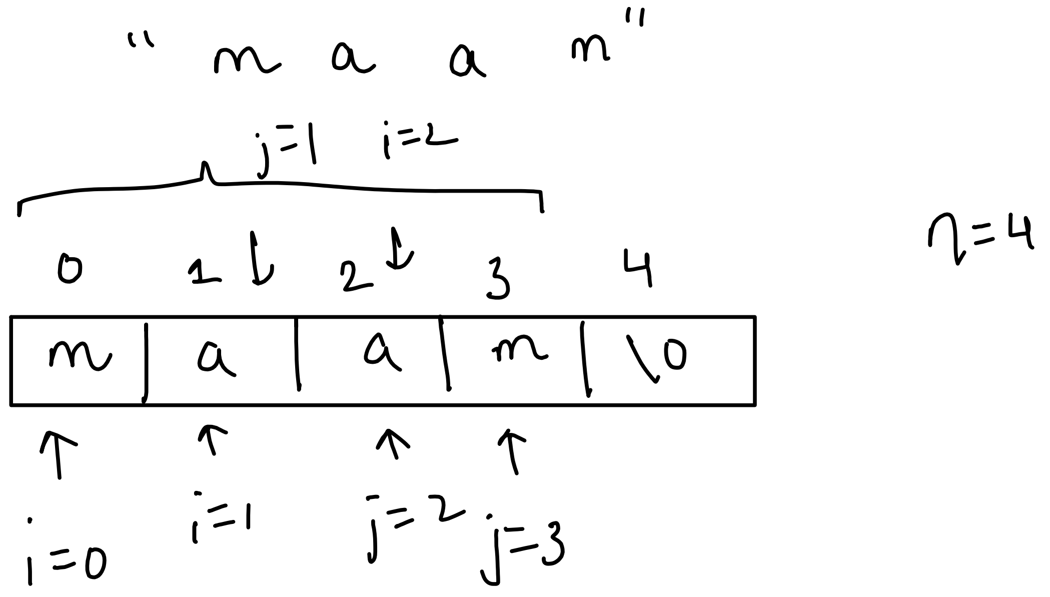
```
if( s[i] == s[j] ) => i++  j--
if (s[i] != s[j] ) => false
```

" r o t a t o r "

$n = 7$

0	1	2	3	4	5	6	7
r	o	t	a	t	o	r	\0

↑ i=0 ↑ i=1 ↑ i=2 ↑ i=3 ↑ j=3 ↑ j=4 ↑ j=5 ↑ j=6


" m a a m"
j=1 i=2
n=4
0 1 2 3 4
m a a m \0
i=0
i=1
j=2
j=3

Check Anagrams

Given a two strings represented as a character arrays, check if they are **anagrams**.

note : assume characters the input strings are lowercase letters (a - z).

Example

Input : "state" and "taste"
Output : true

Input : "abacbac" and "aabbbcc"
Output : false

S1 = state
S2 = taste

s → 1
t → 2
a → 1
e → 1

s → 1
t → 2
a → 1
e → 1

a → 3
b → 2
c → 2

a → 2
b → 3
c → 2

sort aaabbcc

sort aabbbcc

compare
↓ don't match
not angrams

S1 = "state" $\xrightarrow{\text{sort}}$ aestt
S2 = "taste" $\xrightarrow{\text{sort}}$ aestt
comp $\xrightarrow{\text{match}}$ anagrams

1. Sorting Approach (assume len of S1[] and S2[] is equal to n)

a) sort S1[] and S2[] nlogn nlogn

b) compare S1[] and S2[] n

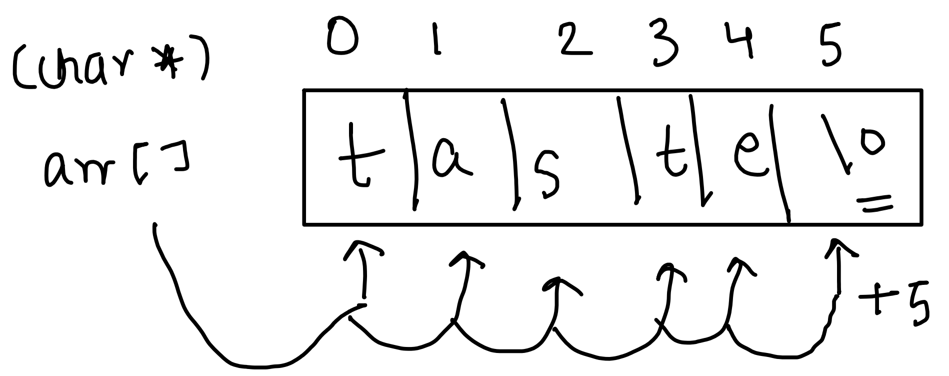
~~2~~nlogn + ~~n~~ ~ O(nlogn)

n = 5

sort(begin, end)

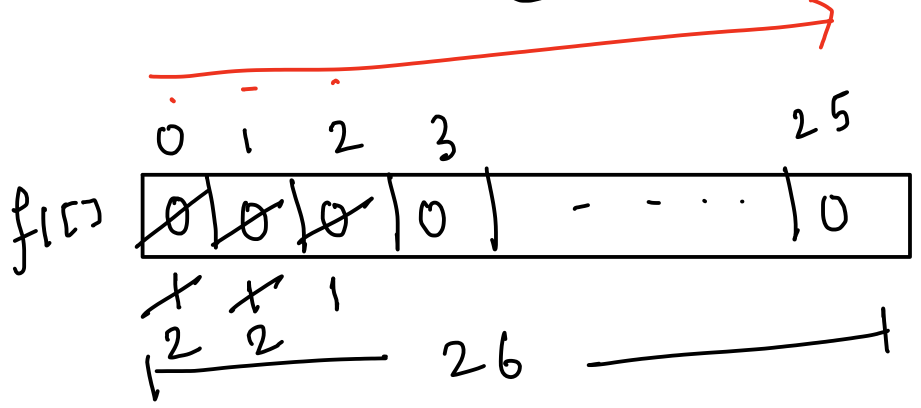
↑ arr   ↑ arr+n

arr[0, n-1]



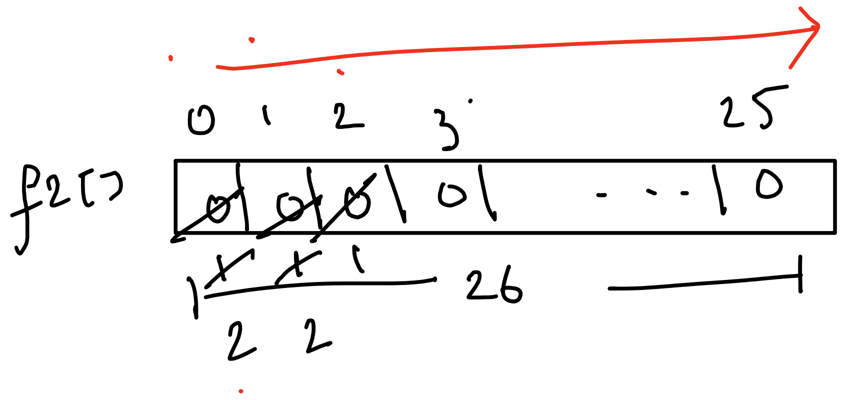
2nd Approach - using freqMaps →

s1[] = "abcab"



0 → a
1 → b
2 → c
⋮
25 → z

s2[] = "cabab"



0 → a
1 → b
⋮
25 → z

ch → idx ?

$$\boxed{ch - 'a' \rightarrow idx}$$

'a'	→	0
'b'	→	1
'c'	→	2
⋮		
'z'	→	25

'a' - 'a' = 97 - 97 = 0

'b' - 'a' = 98 - 97 = 1

'c' - 'a' = 99 - 97 = 2

⋮

'z' - 'a' = 122 - 97 = 25

'7' - '0'

= 55 - 48

= 7 (int)

1st approach

$2n \log_2 n + n$

$\log_a a^b = b$

$n = 4 = 2^2$

$= 2 \cdot 4 \overline{\log_2 2^2} + 4$

$= 8 \cdot 2 + 4$

$= 20$ steps

$n = 8 = 2^3$

$= 2 \cdot 8 \log_2 2^3 + 8$

$= 16 \cdot 3 + 8$

$= 56$ steps

$n = 16 = 2^4$

$= 2 \cdot 16 \log_2 2^4 + 16$

$= 32 \cdot 4 + 16$

$= 144$ steps

$\vdots$

* as n ↑es 2nd apprach becomes better

2nd approach

$n + n + 26$

$= 4 + 4 + 26$

$= 34$ steps

$= 8 + 8 + 26$

$= 42$ steps

$= 16 + 16 + 26$

$= 58$ steps

$\vdots$

Find Largest of N Strings

Given a **N** strings represented as a character arrays, design an algorithm to find the **lexicographically** **largest** string.

Example

Input : "abc", "ae", "xyz", "zz"
Output : "zz"

lsf = "~~abc~~" ~~ae~~ ~~xyz~~ zz

"abc" → lexicographically smallest

"ae"

"xyz"

"zz" → lexicographically largest

```
{
  char str[10];

  str = "abc"; // error
```

String Class in C++

type

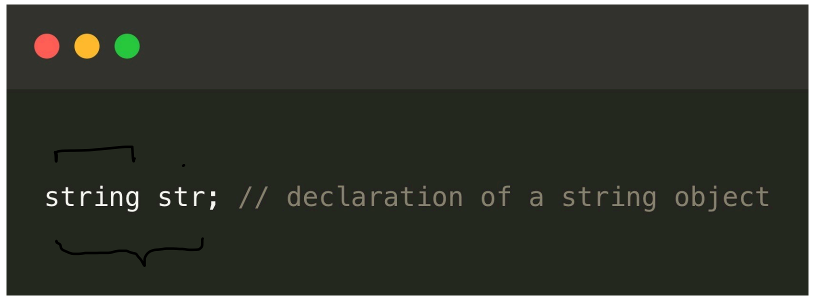
String Class in C++

object → variable

int x;

char ch;

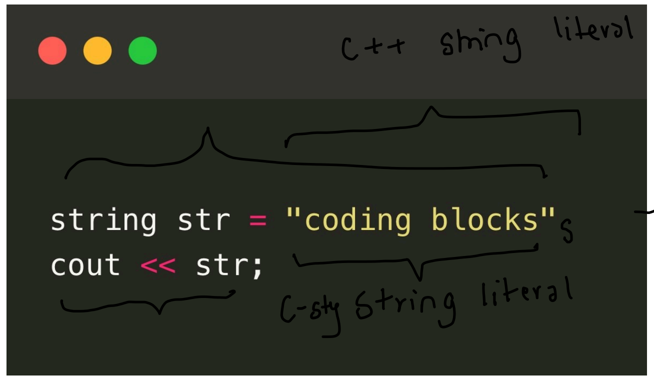
:



```
string str; // declaration of a string object
```

Initialization of a String

int x = 10;



```
string str = "coding blocks"
cout << str;
```

'\0' is added automatically

string str; // declare

str = "wow"; // works

cout << str; // "wow"

char str[100];

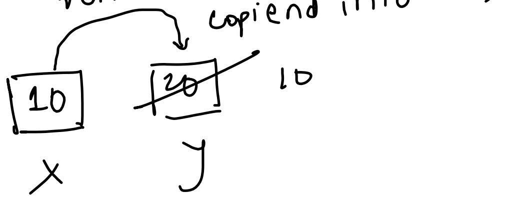
str = "wow"; || error

* "string" is resizable → it can dynamically grow / shrink at runtime

```
string s1 = "abc";
string s2 = "defghi";
s1 = s2;
```

copied into this

```
int x = 10;
int y = 20;
y = x;
```



```
char s1[] = "abc";     → internally 4B are allocated
char s2[] = "defghi"   → internally 7B are allocated

strcpy(s1, s2); // undefined behaviour
```

Indexing a String

```
            0 1 2 3
string s = "a b c d";

cout << s[0]; // a
cout << s[1]; // b
cout << s[2]; // c
cout << s[3]; // d
cout << s[4]; // ? '\0'
```

null char

```
for(int i=0; s[i]!='\0'; i++){
    cout << s[i];
}

for (int i=0; i<n; i++) {
    cout << s[i];
```

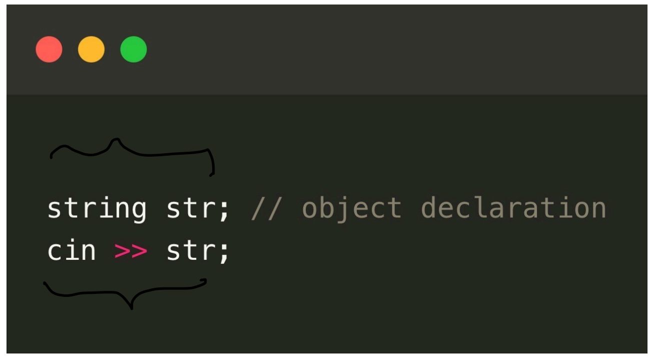
length? → s.size() or s.length()

s.size() → it is a generic fn

s.length() → it is part of the string class

}

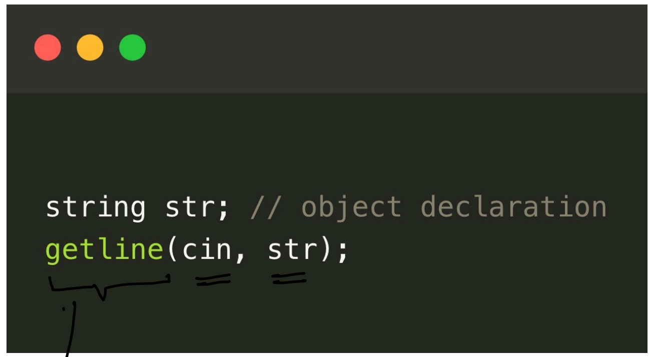
Reading Input into a String



```
string str; // object declaration
cin >> str;
```

cin >> stops reading i/p as soon as it encounters a non-leading whitespace

→ leading whitespaces are ignored by cin>>



```
string str; // object declaration
getline(cin, str);
```

it stops reading as soon as it encounters '\n' ← default delimiting character

↳ to ignore leading whitespaces while using getline, we've to use input manipul. ⇒ ws

getline(cin >> ws, str);

string s = "hello";

cout << s.size(); // 5

size_t (type) → unsigned integers (>, 0)

int n = s.size(); implicit type - casting

max (arg1 (int) , arg2 size_t)

(int) s.size() equivalent static_cast<int