Complete Notes on Strings in C++

---

1. Introduction to Strings

What is a String?

- A string is a sequence of characters.
- In C++, there are **two main types of strings**:
  i. **C-style strings**: Arrays of characters (`char array[]`).
  ii. **C++ strings**: Modern, object-oriented `std::string` from the Standard Template Library (STL).

Difference Between C-Style Strings and `std::string` :

Feature	C-Style String	`std::string`
Declaration	`char str[] = "Hello";`	`std::string str = "Hello";`
Memory Management	Manual (risk of bugs)	Automatic
Functions	Limited (in `<cstring>`)	Rich set of methods in STL
Safety	Error-prone	Safe and modern

---

2. C-Style Strings

Declaration

```
char str[] = "Hello"; // String literal
char str2[10] = "World"; // With specified size
```

Common Functions (in `<cstring>` library)

Function Name	Description	Example
`strlen()`	Returns length of the string	`strlen("Hello") → 5`
`strcpy()`	Copies one string into another	`strcpy(dest, src)`
`strcat()`	Concatenates two strings	`strcat(str1, str2)`

Function Name	Description	Example
`strcmp()`	Compares two strings (`<` , `=` , `>`)	`strcmp("abc", "xyz") → -1`
`strchr()`	Finds first occurrence of a char	`strchr("Hello", 'e') → "ello"`
`strstr()`	Finds substring in a string	`strstr("Hello", "lo") → "lo"`

Example:

```
#include <iostream>
#include <cstring>
int main() {
    char str1[20] = "Hello";
    char str2[20] = "World";

    strcat(str1, str2); // Concatenates str2 to str1
    std::cout << str1 << std::endl; // Output: HelloWorld

    return 0;
}
```

---

3. Modern C++ Strings (`std::string`)

Declaration

```
#include <string>
std::string str = "Hello, World!";
```

Input/Output

1. **Input with `cin` (stops at spaces):**

    ```
    std::string name;
    std::cin >> name; // Input: "John Doe" -> Only "John" is stored
    ```

2. **Input with `getline` (reads full line):**

    ```
    std::string name;
    std::getline(std::cin, name); // Input: "John Doe" -> Stores "John Doe"
    ```

---

4. String Operations

Concatenation

- Use `+` or `+=` operator.

```
std::string s1 = "Hello";
std::string s2 = "World";
std::string result = s1 + " " + s2; // Output: "Hello World"
```

Access Characters

- Use the subscript operator (`[]`) or `.at(index)` .

```
std::string s = "Hello";
std::cout << s[0];       // Output: H
std::cout << s.at(4);    // Output: o
```

Find Substring or Character

- `.find()` returns the index of the first occurrence.

```
std::string s = "Hello";
std::cout << s.find("lo"); // Output: 3
```

Substring

- Use `.substr(start, length)` .

```
std::string s = "Hello";
std::cout << s.substr(1, 3); // Output: ell
```

Length of String

- Use `.length()` or `.size()` .

```
std::string s = "Hello";
std::cout << s.length(); // Output: 5
```

Modifying Strings

Operation	Syntax	Example
Insert	`s.insert(index, substring)`	`"abc".insert(1, "xyz") → "axyzbc"`

Operation	Syntax	Example
Erase	`s.erase(start, length)`	`"abc".erase(1, 1) → "ac"`
Replace	`s.replace(start, len, sub)`	`"abc".replace(0, 2, "xy") → "xyc"`

---

5. String Iteration

Using Loops

```
std::string s = "Hello";
for (char c : s) {
    std::cout << c << " "; // Output: H e l l o
}
```

Using Indices

```
for (size_t i = 0; i < s.length(); ++i) {
    std::cout << s[i];
}
```

---

6. String Comparison

Operators

- Strings can be compared using relational operators (`==` , `<` , `>` , etc.).

```
std::string s1 = "abc", s2 = "xyz";
if (s1 < s2) {
    std::cout << "abc is smaller than xyz";
}
```

---

7. String Functions

Function	Description	Example
`.find(substring)`	Finds first occurrence of substring/char	`"Hello".find("lo") → 3`

Function	Description	Example
`.substr(start, len)`	Returns substring	`"Hello".substr(0, 4) → "Hell"`
`.append(string)`	Appends string	`"Hello".append("World") → "HelloWorld"`
`.insert(index, str)`	Inserts substring at index	`"Hello".insert(2, "abc") → "Heabcllo"`
`.erase(start, len)`	Removes substring	`"Hello".erase(1, 2) → "Hlo"`
`.replace(start, len, s)`	Replaces substring	`"Hello".replace(0, 2, "Hi") → "Hiello"`
`.clear()`	Clears the string	`"Hello".clear() → ""`

---

8. Advanced String Manipulation

Reverse a String

```
std::string s = "Hello";
std::reverse(s.begin(), s.end());
std::cout << s; // Output: "olleH"
```

Count Frequency of Characters

```
#include <map>
std::string s = "abcabc";
std::map<char, int> freq;
for (char c : s) {
    freq[c]++;
}
```

Check for Palindrome

```
#include <algorithm>
std::string s = "madam";
std::string rev = s;
std::reverse(rev.begin(), rev.end());
if (s == rev) {
    std::cout << "Palindrome";
}
```

9. Practical DSA Problems with Strings

Longest Common Prefix

```
#include <vector>
#include <string>
std::string longestCommonPrefix(std::vector<std::string>& strs) {
    if (strs.empty()) return "";
    std::string prefix = strs[0];
    for (int i = 1; i < strs.size(); ++i) {
        while (strs[i].find(prefix) != 0) {
            prefix = prefix.substr(0, prefix.size() - 1);
            if (prefix.empty()) return "";
        }
    }
    return prefix;
}
```

Anagram Check

```
#include <algorithm>
bool areAnagrams(std::string s1, std::string s2) {
    std::sort(s1.begin(), s1.end());
    std::sort(s2.begin(), s2.end());
    return s1 == s2;
}
```

10. Best Practices

- Always prefer `std::string` over C-style strings.
- Use STL functions for efficient string manipulation.
- Avoid hardcoding string sizes; leverage dynamic memory of `std::string`.