



## COMP 2012H Honors Object-Oriented Programming and Data Structures

### Self-study: C++ string class

Dr. Desmond Tsoi

Department of Computer Science & Engineering  
The Hong Kong University of Science and Technology  
Hong Kong SAR, China



## Introduction

- **C strings** consist of a **sequence** of characters terminated by the `'\0'` character, and are stored as **char arrays**.
- **C string** is not a basic data type and does not natively support any operations which are, instead, provided by a set of functions defined in the **cstring** library.
- There is a **C++ string class** which allows you to manipulate **string objects** as if they are from a basic type.
  - ▶ *string construction and initialization*
  - ▶ *input and output*
  - ▶ *indexing*
  - ▶ *comparisons*
  - ▶ *substrings*
  - ▶ *swapping strings*
  - ▶ *string length*
  - ▶ *finding substrings and characters*
  - ▶ *replacing characters in string*
  - ▶ *inserting characters in string*

## Common Public Member Functions

- |                                 |                                          |                                   |
|---------------------------------|------------------------------------------|-----------------------------------|
| • <code>string::append</code>   | • <code>string::end</code>               | • <code>string::operator+=</code> |
| • <code>string::assign</code>   | • <code>string::erase</code>             | • <code>string::operator[]</code> |
| • <code>string::at</code>       | • <code>string::find</code>              | • <code>string::push_back</code>  |
| • <code>string::begin</code>    | • <code>string::find_first_not_of</code> | • <code>string::rbegin</code>     |
| • <code>string::capacity</code> | • <code>string::find_first_of</code>     | • <code>string::rend</code>       |
| • <code>string::clear</code>    | • <code>string::find_last_not_of</code>  | • <code>string::replace</code>    |
| • <code>string::compare</code>  | • <code>string::find_last_of</code>      | • <code>string::resize</code>     |
| • <code>string::copy</code>     | • <code>string::insert</code>            | • <code>string::rfind</code>      |
| • <code>string::c_str</code>    | • <code>string::length</code>            | • <code>string::size</code>       |
| • <code>string::data</code>     | • <code>string::max_size</code>          | • <code>string::substr</code>     |
| • <code>string::empty</code>    | • <code>string::operator=</code>         | • <code>string::swap</code>       |

## Constructors

- Default constructor: To create an empty string.

```
string s; // s is ""
```

- To create a string from  $n$  copies of a character.

```
string rating(3, 'A'); // rating is "AAA"
```

- To create a string object from a C string.

```
string s1("hello");  
string s2 = "hello"; // both are "hello"
```

- To create from a substring of a C string:

```
string s("hello", 4); // s is "hell"
```

## Assignment and Concatenation

- To assign one string object to another string object.

```
string s1("thank"), s2;           // s1 is "thank", s2 is ""
s2 = s1;                          // s2 is "thank" too
```

- To assign a C string to a string object.

```
string s; s = "good";             // s is "good"
```

- To assign a character to a string object.

```
string s; s = 'A';                // s is "A"
```

- To concatenate 2 string objects.

```
string s1("thank"), s2 = "you";   // s1 is "thank", s2 is "you"
string s3;                        // s3 is ""
s3 = s1 + " " + s2;               // s3 is "thank you"
s1 += s2;                         // s1 is "thankyou"
```

## Basic I/O

- To read one word into a string object.

```
string s;
cin >> s;                       // skip leading whitespaces
```

- To output a string object to the console.

```
string s = "hkust";
cout << s;                       // output "hkust" to the screen
```

- To get one line of text until a newline or other delimiter character.

```
string s;
getline(cin, s);                 // read until a newline character
getline(cin, s, 'X');            // read until an 'X' which is not read in
                                // but discarded. Next input starts after 'X'.
```

## Accessing Individual Characters

- To get the number of characters in a string object.

```
string s = "hkust";
cout << s.length();              // output is 5
```

- 2 ways to get the  $j$ th character of a string object.

1. use the subscript operator `[]` like the C strings

- ★  $0 \leq j \leq \text{length}() - 1$
- ★ you don't have to worry about the terminating NULL character, which will be handled by the class.
- ★ if the index  $j > \text{length}() - 1$ , the behaviour is **undefined**.

2. member function `at()`

- ★ similar to `[]`, but check if the index is valid  $\Rightarrow$  **slower**.
- ★ **runtime error** if the index is **beyond** the string length.

```
string s = "hkust";
cout << s.at(1);                 // output is 'k'
s.at(9) = 'a';                  // runtime error
```

## String Comparisons

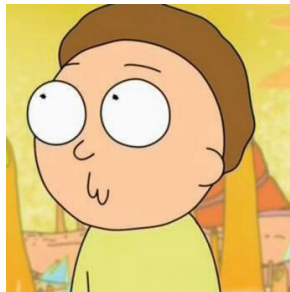
- Comparison is based on the lexicographical order.
- Comparison operators: `<`, `<=`, `>`, `>=`, `==`, `!=`
- compare()** function: `s1.compare(s2)`
  - returns 0 if the strings are the same
  - returns a +ve number if s1 is lexicographically greater than s2
  - returns a -ve number if s1 is lexicographically smaller than s2



## Substrings

- `s.substr(j, N)`: To get a copy of the substring of `s` starting at the index `j`, consisting of `N` characters.

```
// index: 01234567890123456789012345678901234567890123456789
string s = "The rain in Spain stays mainly in the plain.";
string s1 = s.substr(1, 2);
string s2 = s.substr(9, 8);
string s3 = s1 + " is " + s2;
cout << s3 << endl;    // Output: he is in Spain
```



## Find and Replace

- `s.find(s2, j)`: To search for the first occurrence of a substring `s2`, starting from the `j`th character of current string object `s`. It returns
  - if found, the **position** of the found substring in the string object.
  - otherwise, a constant `string::npos` = the max. unsigned int.
- `s.rfind(s2, j)`: similar to `find()` but search backwards.
- `s.replace(j, N, s2)`: To replace the substring of `s` that starts at the `j`th character, consisting of `N` characters by `s2`.



## Example: Find and Replace

```
#include <iostream>    /* File: sfind.cpp */
using namespace std;

int main()
{
    //index: 0123456789012345678901234567890123
    string s("The rain in Spain stays mainly in the plain.");
    cout << s.length() << endl;

    int p1 = s.find("ain");    // Find the first "ain"
    int p2 = s.find("ain", 10); // First "ain" from the 10th char
    int p3 = s.find("ain", 20); // First "ain" from the 20th char
    int p4 = s.find("ain", 30); // First "ain" from the 30th char
    int p5 = s.find("ain", 50); // First "ain" from the 50th char
    cout << p1 << ' ' << p2 << ' ' << p3 << ' ' << p4 << ' ' << p5 << endl;

    s.replace(p1, 3, "hino"); cout << s << endl;
    cout << s.find("ain", 10) << endl; // First "ain" from the 10th char again
    return 0;
}
// 44
// 5 14 25 40 -1
// The rhino in Spain stays mainly in the plain.
// 15
```

## Example: Find and Replace ..

```
#include <iostream>    /* File: find-replace.cpp */
using namespace std;

void print_words(const string word[], int num_words) {
    for (int j = 0; j < num_words; j++)
        cout << word[j] << endl;
}

int main() {
    string s1("et"); // substring to be replaced
    string s2("ad"); // replacing string
    const int num_words = 5;
    string word[num_words] = { "met", "deet", "pets", "eta", "set" };
    cout << "Before replacement" << endl; print_words(word, num_words);

    // Replace the first occurrence of "et" by "ad" in each word
    for (int j = 0; j < num_words; j++)
    {
        int index = word[j].find(s1, 0);
        word[j].replace(index, s1.length(), s2);
    }

    cout << "After replacement" << endl; print_words(word, num_words);
    return 0;
}
```

That's all!  
Any questions?

