70. C++ `string` class

Objectives

- Learn about C strings.
- Learn about the `string` class from the C++ standard library.

Review: C-strings

You already know that a string is an array of `char` terminated by `'\0'`. `'\0'` has numeric value 0. Here's an example:

```
char x[] = "abc"; // same as {'a', 'b', 'c', '\0'}
```

These are called C-strings, i.e., C-strings are character arrays containing the null character.

There are some useful functions to work with C strings of which some are listed below (see CISS240):

```
strcpy(s1, s2) // copy
strcmp(s1, s2) // compare
strcat(s1, s2) // concat
strlen(s1)     // string length
```

Note that in order to work with the above functions, you must include the header `cstring` like so:

```
#include <cstring>
```

Note that the relational operators ==, !=, etc. don't work with the string content but rather with the pointer to the first value the array. In particular, note that `s1 == s2` compares the addresses of `s1` and `s2`. In other words, it does not compare the contents of the arrays. Likewise for `!=`, `<`, etc.

The std::string class in C++

You can use C-strings just fine in C++ but C++ also provides a string class as part of it's standard library that can be easier to work with. The string class in C++ is defined under the std namespace like the rest of the standard library and provides the usual operations you would expect with strings packaged together as a first class object.

Some major features of the C++ string class are outline below:

```
string(const char *) // constructor that lets you
                     // build a C++ string object
                     // from a C-string

=                    // assign a string to another
                     // works even with C strings

==    !=             // compare C++ string objects
>     <              // relate C++ string objects

+=    +              // concatenate string objects

[int]    at(int)     // access characters at a
                     // particular index

substr(int)          // Get substrings at particular
substr(int, int)     // locations in the string

length()             // Get the length of the string

empty()              // Check if the string is empty

swap(string &)       // swap two string values
```

Check C++ references online for more.

Here's a sample main.cpp that shows most of the above in action. Ask yourself what the output should be then verify by running it.

```
#include <iostream>
#include <string>

int main()
{
    std::string s1("Hello, World!"), s2("");
    std::string s3("Hello, World!");
    std::cout << "1:" << s1 << "\n";
    std::cout << "2:" << s1.length() << "\n";
    std::cout << "3:[" << s2 << "]\n";
    std::cout << "4:" << s2.length() << "\n";
    std::cout << "5:" << (s1 != s2) << "\n";
    std::cout << "6:" << (s1 == s2) << "\n";
    std::cout << "7:" << (s2.empty()) << "\n";
    std::cout << "8:" << (s2 == "") << "\n";
```

```
    s2 = "!!";
    std::cout << "9:" << s1 + s2 << "\n";
    std::cout << "10:" << s1 + "!!" << "\n";
    s1 += s3;
    std::cout << "11:" << s1 << "\n";
    s1 += "???";
    std::cout << "12:" << s1 << "\n";
    std::cout << "13:" << s1.substr(3) << "\n";
    std::cout << "14:" << s1.substr(3, 1) << "\n";
    std::cout << "15:" << s1[2] << "\n";
    s1[2] = 'X';
    std::cout << "16:" << s1 << "\n";

    return 0;
}
```

Searching for substrings

You can use the methods `find()` and `rfind()` provided by the string library to search for substrings within a `string` object. Use `find()` to search from the left side of the string and `rfind()` to search from the right side instead. You can use both C++ `string` objects and C strings as the substring values to search for.

Example.

```
std::string s1 = "01234567890123456789";
std::string s2 = "23";
std::cout << "1." << s1.find(s2) << "\n";
std::cout << "2." << s1.find("23") << "\n";
std::cout << "3." << s1.find("$") << "\n";
std::cout << "4." << s1.rfind("23") << "\n";
```

Both `find()` and `rfind()` return the index where the substring first occurs. If the substring isn't found, `std::string::npos` is returned which is the same as -1 but as an `unsigned int`.

Searching for characters

The standard string library provides the following methods to search for characters in a string:

- find_first_of() which takes in a character, a C-string or a string object as its argument and finds the first index from the left side in the invoking string object where any of the characters in the argument occur.
- find_last_of() which is similar to find_first_of() except it searches from the right side instead.
- find_first_not_of() which also takes in a character, a C string or a string object as argument but finds the index of the first character in the invoking string object that isn't also in the argument provided to the method.
- find_last_not_of() which is similar to find_first_not_of() except that it searches from the right side instead.

Example.

```
std::string s = "01234567890123456789";
std::string t = "23";
std::cout << "1." << s.find_first_of(t) <<"\n";
std::cout << "2." << s.find_first_not_of("01") << "\n";
std::cout << "3." << s.find_last_of("012") << "\n";
std::cout << "4." << s.find_last_not_of("012") << "\n";
```

As with substring search, if nothing is found, std::string::npos is returned which is the same as -1 but as an unsigned int.

Replacement

You can use the `replace()` method to replace the contents of a string with the content of another. You need to specify indices to replace for the `string` object invoking the method but if you want, you can also specify the portion of the string passed in as the argument to use for the replacement.

Example.

```
#include <iostream>
#include <string>

int main()
{
    std::string s1("0123456789");
    std::string s2("abcdef");

    s1.replace(3, 2, s2);
    std::cout << s1 << "\n";

    s1 = "0123456789";

    s1.replace(3, 2, s2, 1, 3);
    std::cout << s1 << "\n";

    return 0;
}
```

There are still other forms of `replace()` available which can take in C strings or characters or even let you repeat a character a certain number of times. Research using Google when you have free time.

Insertion

You can use the `insert()` method to insert the contents of a string into another. You need to specify the index of the invoking `string` object to insert at but you can also specify the portion of the string passed in as the argument to insert.

Example.

```
#include <iostream>
#include <string>

int main()
{
    std::string s1("0123456789");
    std::string s2("abcdef");

    s1.insert(3, s2);
    std::cout << s1 << "\n";

    s1 = "0123456789";
    s1.insert(3, s2, 2, 3);
    std::cout << s1 << "\n";

    return 0;
}
```

Even more method signatures of `insert()` exist as well.

Type conversion

Given a C++ string object, you can get a C string with the same content using the `c_str()` method.

Example.

```
#include <iostream>
int main()
{
    std::string s1 = "Hi!";
    const char * s2;
    s2 = s1.c_str();
    std::cout << s2 << "\n";

    return 0;
}
```

You can use a regular constant character pointer to bind to the C-string returned but you **shouldn't delete the pointer** if you do since the pointer will be pointing to the C-string content of the C++ `string` object.

```
const char * s2;
s2 = s1.c_str();
std::cout << s2 << "\n";
delete [] s2; // BAD!!!
```

Since the C-string returned is a constant pointer to a character array, you **can't change its content** through simple assignment.

```
const char * s2;
s2 = s1.c_str();
s2[1] = '$'; // BAD!!!
```

External Resources

There are many resources available to you to learn more if you are interested.

Most textbooks contain some information on the `std::string` class.

The internet is full of websites containing a detailed reference to the C++ standard library. The following are two good ones, just Google for more:

- http://www.cplusplus.com
- http://en.cppreference.com

You should get used to researching content online. When you have questions, you are more than likely running into an issue that other people also have run into in the past. Leverage their experience. Find and join a good C++ forum. At the very least, join the following discussion websites meant for programmers:

- http://stackoverflow.com/
- http://programmers.stackexchange.com/

Of course you shouldn't outright plagiarize content you find online. It won't really help you much to just copy-paste code in anyway.

Exercises

Exercise. Do an experiment to verify that `operator>>` (i.e., input) is overloaded to work with `std::string` objects. Note that a whitespace will terminate an input.

Exercise. Write a program that does the following: Declare a `std::string` object and initialize it with a string. Create an array of `std::string` objects from the words in the first string. The separator of words include the space and the period. Print all the words in the first string, each on a separate line. Print the total number of words and the average number of characters per word.

Exercise. Write a program that prompts the user for a string and prints "integer" if the string is made up of digit characters where the first is not '0'; otherwise the program prints "not integer".

Exercise. Write a function that accepts a `std::string` object and returns the string with words capitalized. In other words, if the object passed in is `std::string("hello world")`, then `std::string"Hello World")` is returned; if the object passed in is `std::string("ok. that's great!")`, then `std::string("Ok. That's great!")` is returned. Assume that the only word separators are the space, the period, and the exclamation mark.