Strings

Strings

We've been using strings a little bit in our programs so far,

But we haven't looked at a lot of the things that C++ strings can do.

The C++ string is an **object** that has a set of functionality that can be really handy!

String size

We can get the size of the contents of a string by using the **`.size()`** function.

```
string sentence;
cout << "Enter a sentence: ";
getline( cin, sentence );

cout << "Length is: " << sentence.size() << endl;
```

```
Enter a sentence: the quick brown fox jumps over the lazy dog
Length is: 43
```

Iterating through strings

The square brackets,
[ ]
is the **subscript operator** in C++.

```
string word;
cout << "Enter a word: ";
cin >> word;

for ( int i = 0; i < word.size(); i++ )
{
    cout << word[i] << endl;
}
```

We can get single **char**s from a string by using it like an array.

```
Enter a word: hello
h
e
l
l
o
```

Finding substrings

We can use the **`find`** function to find a substring within a string.

`find` returns the **position** that the (first) substring is found.
If no substring is found, it is set to the value of **`string::npos`**.

```
size_t position = phrase.find( word );

if ( position == string::npos )
{
    cout << "Word was not found." << endl;
}
else
{
    cout << "Word was found at position: "
        << position << endl;
}
```

Note that
size_t
Is an alias of
an **unsigned integer**.

Finding substrings

```
string phrase = "the hamburgers are not made from ham.";

cout << phrase << endl;
cout << "Search for what? ";
string word;
cin >> word;

size_t position = phrase.find( word );

if ( position == string::npos )
{
    cout << "Word was not found." << endl;
}
else
{
    cout << "Word was found at position: "
        << position << endl;
}
```

Only the position of the first found match is returned.

```
Search for what? ham
Word was found at position: 4
```

```
Search for what? meow
Word was not found.
```

String Replace

We can replace portions of a string with the **`replace`** function.

We pass in the **position** and the **length** that we want to replace, and the **string** of what it should be replaced with.

```
0   S
1   t
2   a
3   r
4
5   T
6   r
7   e
8   k
9
10  i
11  s
```

“Trek” begins at position **5** and is **4** characters long.

```
string statement = "Star Trek is best.";
cout << statement << endl;
statement = statement.replace( 5, 4, "Wars" );
cout << statement << endl;
```

```
Star Trek is best.
Star Wars is best.
```

String Find and Replace

We can combine the **find** and **replace** function to write a function that will swap out one string with another string.

```
string FindAndReplace( string& original, string& findme, string& replaceme )
{
    size_t pos = original.find( findme );
    if ( pos != string::npos )
    {
        string fixed = original.replace( pos, findme.size(), replaceme );
        return fixed;
    }
    return original;
}

int main()
{
    string sentence = "Class is Monday, Wednesday, and Friday at 5:30";
    string badText = "Monday, Wednesday, and Friday";
    string goodText = "Tuesday and Thursday";
    string fixed = FindAndReplace( sentence, badText, goodText );
    cout << fixed << endl;

    return 0;
}
```

String Find and Replace

We can combine the **find** and **replace** function to write a function that will swap out one string with another string.

```
string FindAndReplace( string& original, string& findme, string& replaceme )
{
    size_t pos = original.find( findme );
    if ( pos != string::npos )
    {
        st
        re
    }
    return
}

int main()
{
    string sentence = "Class is Monday, Wednesday, and Friday at 5:30";
    string badText = "Monday, Wednesday, and Friday";
    string goodText = "Tuesday and Thursday";
    string fixed = FindAndReplace( sentence, badText, goodText );
    cout << fixed << endl;

    return 0;
}
```

stuff

```
Class is Tuesday and Thursday at 5:30

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Substring

We can get a substring from a string with the **`substring`** function, which requires a starting position and length.

```
string sentence = "One plus two plus three";
string one = sentence.substr( 0, 3 );
string two = sentence.substr( 9, 3 );
string three = sentence.substr( 18, 5 );

cout << one << endl;
cout << two << endl;
cout << three << endl;
```

```
One
two
three
```

Get a line

`getline` is not a function that belongs to the **string** class, but it can be used to get a line of text from the user and store it in a **string**.

```
string name;
cout << "What is your name? ";
getline( cin, name );
cout << "Hello, " <<  name << endl;
```

```
What is your name? Muper Sario
Hello, Muper Sario
```

Get a line

Note that if you're inter-mixing your input methods, you might need to use **`cin.ignore()`** before you change, or you might get wonky results:

```
string name;
string resume;

cout << "Continue? (y/n): ";
cin >> resume;

if ( resume == "y" )
{
    cout << "What is your name? : ";
    getline( cin, name );

    cout << "Hello, " << name << endl;
}

cout << "The end" << endl;
```

```
Continue? (y/n): y
What is your name? : Hello,
The end
```

It totally skipped the getline statement!

Get a line

Note that if you're inter-mixing your input methods, you might need to use **`cin.ignore()`** before you change, or you might get wonky results:

```
string name;
string resume;

cout << "Continue? (y/n): ";
cin >> resume;

if ( resume == "y" )
{
    cout << "What is your name? : ";
    cin.ignore();
    getline( cin, name );

    cout << "Hello, " << name << endl;
}

cout << "The end" << endl;
```

```
Continue? (y/n): y
What is your name? : Guybrush
Hello, Guybrush
The end
```

Now it's OK!

Strings

You can find out more about the **string** class and its functions at http://www.cplusplus.com/reference/string/string/

std::string

<string>

```
typedef basic_string<char> string;
```

String class

Strings are objects that represent sequences of characters.

The standard `string` class provides support for such objects with an interface similar to that of a standard container of bytes, but adding features specifically designed to operate with strings of single-byte characters.

The `string` class is an instantiation of the basic_string class template that uses `char` (i.e., bytes) as its *character type*, with its default char_traits and allocator types (see basic_string for more info on the template).

Note that this class handles bytes independently of the encoding used: If used to handle sequences of multi-byte or variable-length characters (such as UTF-8), all members of this class (such as length or size), as well as its iterators, will still operate in terms of bytes (not actual encoded characters).

Member types

member type	definition
value_type	char
traits_type	char_traits<char>
allocator_type	allocator<char>
reference	char&
const_reference	const char&
pointer	char*
const_pointer	const char*
iterator	a random access iterator to char (convertible to const_iterator)
const_iterator	a random access iterator to const char
reverse_iterator	reverse_iterator<iterator>

Strings

Most of the pages about functions will have a parameter list and sample code.

public member function

std::string::find

<string>

C++98 C++11

string (1)	`size_t find (const string& str, size_t pos = 0) const;`
c-string (2)	`size_t find (const char* s, size_t pos = 0) const;`
buffer (3)	`size_t find (const char* s, size_t pos, size_t n) const;`
character (4)	`size_t find (char c, size_t pos = 0) const;`

Find content in string

Searches the string for the first occurrence of the sequence specified by its arguments.

When *pos* is specified, the search only includes characters at or after position *pos*, ignoring any possible occurrences that include characters before *pos*.

Notice that unlike member find_first_of, whenever more than one character is being searched for, it is not enough that just one of these characters match, but the entire sequence must match.

Return Value

The position of the first character of the first match.
If no matches were found, the function returns string::npos.

size_t is an unsigned integral type (the same as member type `string::size_type`).

Example

```
// string::find
#include <iostream>       // std::cout
#include <string>         // std::string

int main ()
```