[Next](http://www.oopweb.com/CPP/Documents/CPPHOWTO/Volume/C++Programming-HOWTO-8.html) [Previous](http://www.oopweb.com/CPP/Documents/CPPHOWTO/Volume/C++Programming-HOWTO-6.html) [Contents](http://www.oopweb.com/CPP/Documents/CPPHOWTO/Volume/C++Programming-HOWTO.html#toc7)

## 7. The Standard C++ Library string class

While the previously mentioned String class (note the uppercase S), is a good thing for people coming from Java, then you should take notice of the "real" string class provided by The Standard C++ Library.

The string class was made to overcome one of the greatest pitfalls in C; character arrays. While character arrays are extremely fast, they have many bad sides. Character arrays are the cause of many bugs, and parsing character arrays is very time consuming.

The string class brings a good interface for parsing and handling strings, and it's even STL compatible, so it can be used with all the general STL algorithms. Actually you could say that a string is a **vector<char>**. A container of chars, or an advanced array of chars.

Useful string references can be found at the following sites:

* SGI STL basic\_string reference: <http://www.sgi.com/tech/stl/basic_string.html>.

## 7.1 string by example

Creating a string is easy:

#include <string>  
#include <iostream>  
  
using namespace std;  
  
int main()  
{  
 string str("Hello World!"); // Or string str = "Hello World!";  
 cout << str << endl;  
}

This code will create a string called "**str**', and put "**Hello World!**' into it. It is then being outputted to standard output by using cout.

(Note that I will skip the headers and the namespace from now on.)

Taking a substring of a string is also easy:

string str("Hello Universe!");  
string start = str.substr(0, 5);  
string end = str.substr(5);

This will put the first 6 characters into the string "**start**", and the rest into "**end**".

To get the size or length of a string, you would simply do this:

string str("How long is this string?");  
cout << "Length of string is: " << str.size() << endl;

You can also use **length()** which works exactly the same.

## 7.2 Searching a string

Searching a string is much easier than using plain character arrays, the string class provides efficient member functions to search through the string. All member functions return string::size\_type.

**string search member functions**

| Member function | Purpose |
| --- | --- |
| find() | find the first position of the specified substring |
| find\_first\_of() | equal to find(), but finds the first position of any character specified |
| find\_last\_of() | equal to find first of(), but finds the last position of any character specified |
| find\_first\_not\_of() | equal to find first of(), but returns the position of the first character not of those specified |
| find\_last\_not\_of() | equal to find last of(), but returns the last position of any characters not specified |
| rfind() | equal to find(), but searches backwards |
|  |

A very common thing to do, is to search a string for contents. This can be done by using **find()**

string str("Hello, can you find Ben?");  
string::size\_type position = str.find("Ben");  
cout << "First occurrence of Ben was found at: " << position << endl;

This code makes a case sensitive search for '**Ben**' in the string, and puts the start position in the variable '**position**' of type string::size\_type. Note that the return value is not an int, but a string::size\_type which is a special implementation defined integral value.

The member function **find\_first\_of()** needs a practical introduction, consider this:

string s = "C++ is an impressive language.";  
string::size\_type pos = s.find\_first\_of(" .");  
  
while (pos != string::npos) {  
 cout << "Found space or dot at: " << pos << endl;  
 pos = s.find\_first\_of(" .", pos + 1);  
}

By using **find\_first\_of()**, we can search the string for **any** character of the first argument, here we decide to search for a space or a dot.

Try compiling the program and check the output.

## 7.3 A string tokenizer

A very common operation with strings, is to tokenize it with a delimiter of your own choice. This way you can easily split the string up in smaller pieces, without fiddling with the find() methods too much. In C, you could use **strtok()** for character arrays, but no equal function exists for strings. This means you have to make your own. Here is a couple of suggestions, use what suits your best.

The advanced tokenizer:

void Tokenize(const string& str,  
 vector<string>& tokens,  
 const string& delimiters = " ")  
{  
 // Skip delimiters at beginning.  
 string::size\_type lastPos = str.find\_first\_not\_of(delimiters, 0);  
 // Find first "non-delimiter".  
 string::size\_type pos = str.find\_first\_of(delimiters, lastPos);  
  
 while (string::npos != pos || string::npos != lastPos)  
 {  
 // Found a token, add it to the vector.  
 tokens.push\_back(str.substr(lastPos, pos - lastPos));  
 // Skip delimiters. Note the "not\_of"  
 lastPos = str.find\_first\_not\_of(delimiters, pos);  
 // Find next "non-delimiter"  
 pos = str.find\_first\_of(delimiters, lastPos);  
 }  
}

The tokenizer can be used in this way:

#include <string>  
#include <algorithm>  
#include <vector>  
  
using namespace std;  
  
int main()  
{  
 vector<string> tokens;  
  
 string str("Split me up! Word1 Word2 Word3.");  
  
 Tokenize(str, tokens);  
  
 copy(tokens.begin(), tokens.end(), ostream\_iterator<string>(cout, ", "));  
}

The above code will use the Tokenize function, take the first argument **str** and split it up. And because we didn't supply a third parameter to the function, it will use the default delimiter " ", that is - a whitespace. All elements will be inserted into the vector **tokens** we created.

In the end we copy() the whole vector to standard out, just to see the contents of the vector on the screen.

Another approach is to let stringstreams do the work. streams in C++ have the special ability, that they read until a whitespace, meaning the following code works if you only want to split on spaces:

#include <vector>  
#include <string>  
#include <sstream>  
  
using namespace std;  
  
int main()  
{  
 string str("Split me by whitespaces");  
 string buf; // Have a buffer string  
 stringstream ss(str); // Insert the string into a stream  
  
 vector<string> tokens; // Create vector to hold our words  
  
 while (ss >> buf)  
 tokens.push\_back(buf);  
}

And that's it! The stringstream will use the output operator (>>) and put a string into **buf** everytime a whitespace is met, **buf** is then used to push\_back() into the vector. And afterwards our vector **tokens** will contain all the words in **str**.

[Next](http://www.oopweb.com/CPP/Documents/CPPHOWTO/Volume/C++Programming-HOWTO-8.html) [Previous](http://www.oopweb.com/CPP/Documents/CPPHOWTO/Volume/C++Programming-HOWTO-6.html) [Contents](http://www.oopweb.com/CPP/Documents/CPPHOWTO/Volume/C++Programming-HOWTO.html#toc7)