String Class

Strings are much safer than C-strings. One benefit to using strings is that you do not have to worry about the **null zero** like you do with C-strings. Additionally, you also do not need to worry about the size of your string. For example, you are able to concatenate two strings of any length at any time. Strings are not a built-in type like **int** and **char**. Therefore, you need to include the preprocessor directive **#include <string>**.

There are two ways to declare/initialize a string on the same line:

```
string myString("This is a string");
string myString2 = "String";
```

Input/Output

There are two ways to obtain strings from user input. The first way is the same as any other input. If you go this route then input stops at any whitespace. That is, what will be placed in the string variable is one word without whitespace. The second way is to use the **getline()** function. This function takes up to three arguments. The first argument is where the input is coming from (cin or from a file), the second argument is the string variable for the string to be put into, and the third argument is a character delimiter (where to stop the string). The last argument is optional. If it is omitted, then the string input will stop at the first newline (where you hit enter). This is useful if you want more than one word in your string. The example program **InputString.cpp** shows three input examples.

To output strings you would use the same method as any other variable. For example:

```
string string1 = "Hello";
cout << string1 << " World"; //Space before W to keep words separated
```

Features

A convenient feature with strings is that you are able to use some binary operators with them (**=**, **+**, **+=**, **==**, **etc..**). Here are examples of each of the aforementioned binary operators:

```
myString = "Turtle"; //Changed above string to Turtle
string anotherString = "Hi " + myString; //anotherString is Hi Turtle
string newString = myString + myString2; //newString is TurtleString
myString += myString2; //myString is now TurtleString
if(newString == myString) //Boolean check for equality
    cout << "newString and myString are equal"; //They are equal
```

As you may notice, unlike C-strings, you are able to re-assign the value of your string at your leisure with the equals operator. For concatenation, you are able to mix string literals and string variables. The shorthand way of concatenation is using the plus/equals operator. This works the same way as if you were to use it on an **int**. Lastly, you are able to check to see if two strings are equal lexicographically. Since this is done lexicographically, the other boolean operators are able to be used.

Another convenient aspect of strings is that they are able to be accessed element-by-element just like an array. This also lets you alter an individual indexed element just as you could with a C-string. Remember the first index is always zero. Here is an example:

```
String1[0] = 'h'; //Changes Hello to hello
```

You are able to store a C-string into a string quite conveniently:

```
char cString[] = "C-String";
string s1 = cString;
```

Storing a string into a C-string is a bit different. You need to use the **strcpy()** function, which is in the **#include <cstring>** preprocessor directive. The first argument is your C-string variable and the second argument is your string variable. Another thing to note is you have to use the member function **.c_str()** appended to your string variable. Be conscious about the size of your C-string and make sure to leave room for the **null zero**. For example:

```
char anotherCString[10];
string s2 = "String";
strcpy(anotherCString, s2.c_str());
```

Since strings are a class, there are many member functions you can use to manipulate and find out information in regards to your string. For example, you would not use the **sizeof()** function to find out the size of your string. You would use the member function **.length()** appended to your string variable. Examples of some of the most common member functions are shown in **StringMemberFunctions.cpp** with explanations on the next page (**String Member Functions**). For a more comprehensive list, visit **http://www.cplusplus.com/reference/string/string/**