COMP161 - Lab 7

Spring 2014

For this lab you'll be exploring parts of the C++ *string* library in order to get comfortable working with Object and array types.

Write your own tests

Reading the reference for a library of code is often step one in learning how to use the library. However, you must, at some point put your theoretical knowledge of the library to the test by actually using the code. It's often better to do this outside of your intended application. In our case, we'd like to know the ins and outs of strings before we write an obviously string based program: hangman. A simple way to get at the library without worrying about bigger applications is to *write your own set of tests for the code you intend to use*. This simple exercise let's you verify that you know how to invoke the code and that your expectations for the code match the reality.

Your goal for the day is to get comfortable with the C++ string library and in particular the string class[1]. To accomplish this goal you'll first need to read the documentation for a specific procedure and then write some gTest unit tests[2] for that procedure to test and demonstrate your understanding of the documentation. At the end of lab, submit the file with your tests as *lab8* using *handin*. Your tests should all be new and just copies of test cases found on cplusplus.com.

[1] The reference documentation can be found on `http://www.cplusplus.com/reference/string/`

[2] You'll want these `https://code.google.com/p/googletest/wiki/Primer#String_Comparison`

String Class Methods

Most of the procedures we need are actually OBJECT METHODS for the C++ string class[3]. As we discussed in class, this means we use the *dot operator* and call the method relative to a specific string as opposed to passing that string to the procedure as an input. This also means the methods must be invoked on a *string variable* and not a *string literal*. The string class is documented here: `http://www.cplusplus.com/reference/string/string/`.

[3] recall: a class is a non-primitive type

Strings as Recursive Objects

As we know from working with Racket lists, working with recursive data requires three things: the ability to recognize the non-recursive empty case, the ability to select the first of an non-empty recursive data structure, and the ability to select the rest of a non-empty, recursive data structure. It is often advantageous to treat string data

as a recursive data structure. The string library gives us everything we need for this. Write a TEST block for each of the following bullet points:

- EMPTY Write tests to demonstrate how you can the method *empty*, *size*, or *length* to see if a string is empty or not[4].
- FIRST Write tests to demonstrate how you can use the method *at* or the operator *[]* to select the first, location 0, character of a string.
- REST Write tests to demonstrate how you can use the method *substr* to get the rest (all but the first) of a string.
- LAST Write tests to demonstrate how you can use *at* or *[]* to select the last character in the string.
- BUTLAST Write tests to demonstrate how you can use *substr* to select all but the first character of a string
- *LeftHalf* Write tests to demonstrate how you can use *substr* to select the left half of a string.
- *RightHalf* Write tests to demonstrate how you can use *substr* to select the right half of a string.[5]

[4] "" is the empty string literal

[5] Be certain your left and right halves should combine back to the whole, original string.

As some of the cases above imply, recursion doesn't have to happen in the first+rest fashion you're used to. Tuck that thought away for later.

Comparing Strings

The comparison of strings can seem a bit quirky at first. There are two ways to compare strings: the *compare* method and overloaded *relational operators*[6]. Write a TEST block for each of the following bullets. Use a mixture of *compare* and the relational operators.

[6] == , !=, <=,>=,<,>

- Write tests to demonstrate that string comparison is case-sensitive.
- Write tests to demonstrate what comes first, lower or upper case[7]

[7] i.e. demonstrate the exact effect of cases on string comparison

Searching Strings

The string class provides a robust set of methods for finding strings within strings. Write a TEST block for each of the following bullets.

- Write tests to demonstrate the use of *find* to determine the location of the first occurrence a specific letter in a string. Now find a string of length greater than 1.
- Write tests to demonstrate the use of *find* to determine that a specific letter is not in a string. Now detect the absence of a string of length greater than 1.

Mutator methods

So far we've looked at accessors or predicates; none of the methods we've used are string mutators. Let's return to some of the recursion logic and consider a mutation based approach. Write a TEST block for each of the following bullets.

- Write tests to demonstrate the use of *erase* to change a string to its rest (all but the first).
- Write tests to demonstrate the use of *erase* to change a string to its butLast (all but the last).

Remember that strings have two key data properties: the character sequence and the size. So, the expected change should be reflected in both of these properties.

getline

By default, streaming input reads everything up to the first whitespace character. Sometimes you want to get a whole line, i.e. everything up to the "enter key" character. The string class provides a method specifically for doing this. Write a TEST block for each of the following bullets.

- Write tests to demonstrate the use of *getline* to read an entire line of input from the standard input.

Library Procedures for Number to String

We often encounter string representations of numbers where we need actually numbers[8]. Thankfully, the string library[9] provides some convenient procedures for managing these conversions. Write a TEST block for each of the following bullets.

[8] or numbers where we need strings

[9] not class

- Write tests to demonstrate the use of *to_string* to convert numerical values to strings
- Write tests to demonstrate the use of *stoi* to convert ints to strings
- Write tests to demonstrate the use of *stod* to convert doubles to strings

More?

At this point you're done. Submit your tests. However, consider exploring the following tasks.

- All of the methods you tested are accessors or predicates. Consider re-visiting the recursive procedures[10] using the string mutator *erase*.
- You know we're going to write a hangman game. So if you see a method or procedure that seems useful in that context, test it out.
- You might see if you can figure out how to use *find* to find *all* the locations of a specific letter[11].
- Another usefully exercises is to attempt to write your own versions of these methods/procedures using only basic C++ and *length* and *at*[12] methods.[13]

[10] rest,butLast,leftHalf,rightHalf

[11] hint: use a loop

[12] or []

[13] now that you really know *what* these things are and what they do, deepen your understanding by figuring out *how* they get it done.