COMP161 - Lab 6 & Homework 6

Spring 2016

For this lab you'll work on writing a mutator procedure. For homework you'll work on an output procedure. Both topics are covered in lecture notes 9.

Lab 6

An n-gram letter frequency analysis counts the frequency of n letter sequences in a string. These statistics can then be used for several purposes in natural language processing and cryptography. A common technique to handle the begging and end of a document is to pad both with $(n-1)$ special characters such that the first n gram is the first letter of the text preceded by $(n-1)$ special characters and the last n gram is the last character followed by $(n-1)$ special characters. It is usually important that the padding character is unique and doesn't occur elsewhere in the text as it signifies the beginning and end of the document.

Your task is to design and develop a general purpose STRING MUTATOR called *add_padding* that can be used for this purpose. The procedure should take a string *str*, a positive integer *pad_size*, and a character *pad_char*. If the character *pad_char* doesn't occur in *str*, then it should then modify *str* by adding *pad_size* occurrences of *pad_char* to both the beginning and end of the string. If *pad_char* does occur in *str*, then it should leave *str* unchanged.

In order to complete this procedure, you'll need to make use of the *fill constructor*[1] for the std::string class as well as the std::string method *find*[2]. So, a secondary objective of this lab is to give you practice reading a language reference. As always, your procedure should be fully documented[3] and have a sufficient set of tests in addition to be fully defined. At the end of lab, submit a compilable set of code as *lab7* using the *handin* command.

[1] http://www.cplusplus.com/reference/string/string/string/

[2] http://www.cplusplus.com/reference/string/string/find/

[3] now that we're writing procedures for effect, don't forget post conditions

Homework 6

Due By 8 am, Monday March 2nd

Printing data in an organized, tabular form is a common output task. Tabular output presents data aligned into columns and rows and typically prints each value in a specific way. For this problem you are to write an output procedure called *styleRowOut* that produces *one row* of a table of beer style statistics like those see here:

```
Name         OG      FG      ABV    IBU    SRM
Maibock      1.07    1.016   7.2    29     7
Bock         1.07    1.021   7      25     25
Doppelbock   1.077   1.017   7.3    22     21
Eisbock      1.095   1.028   11.5   30     34
```

The first row you see is the header; your procedure does not print that row. The remaining rows are the data rows and are examples[4] of the kind of output your procedure should produce. Each row contains a string, the name, and five doubles.

[4] free test cases!

- The *OG* and *FG*[5] are measurements of beer taken before and after fermentation. They're always reported to three decimal places and the values fall between 1.0 and 2.0.

[5] Original and Final Gravity

- The *ABV*[6] measures the strength of the beer. It is always reported to one decimal place. The ABV tends to fall between be between 3 and 15.

[6] Alcohol by Volume

- The *IBU*[7] measures the hop character and is reported with zero decimal places. The IBUs of a beer tend to be between 10 and 120.

[7] International Bittering Units

- The *SRM*[8] is a measure of beer color and is reported with zero decimal places. The SRM of a beer is typically between 5 and 50.

[8] Standard Reference Measure

The *iomanip*[9] provides a few key manipulators for controlling how output streams print values. For this lab we're concerned with the way in which a double is printed and the general width, i.e. number of characters, values take up in genreal. The *std::setprecision*[10] manipulator lets your determine the number of decimal places printed. Be aware that this causes rounding off of values. If you want to guarantee that the precision is exactly the number of places printed, i.e. if you want trailing zeros, then you need to use *std::fixed*[11] from *iostream*. So, by combining std::precision and std::fixed we can make something like 1.1 print like 1.100.

[9] http://www.cplusplus.com/reference/iomanip/

[10] http://www.cplusplus.com/reference/iomanip/setprecision/

[11] http://www.cplusplus.com/reference/ios/fixed/

The *std::setw* [12] manipulator lets you specify the width of a printed value and is crucial for building a column in a table. Every column entry should print to the same number of characters. If you set the

[12] http://www.cplusplus.com/reference/iomanip/setw/

printing width to 10, then print 2.3 the stream will take care of padding the output with 7 spaces to fill the required 10 character width. By default, padding done to fill the width is done on the left. This results in right justified data. You can change this to left justification by using *std::left*[13]. There's also a *std::right* and a *std::internal* that lets you specify an arbitrary fill point.

[13] http://www.cplusplus.com/reference/ios/left/

To build our table we want print values left-justified with the following widths:

- Names should be printed 20 wide.
- All the numerical values should be printed 8 wide.

It should be noted that the example table shown above is not necessarily printed with these width values. It was generated by Mathematica and not a C++ program.