CSCE 2014 – Programming Project 4

Midpoint Due Date – Mar 17, 2017 at 11:59pm
Final Due Date – Mar 31, 2017 at 11:59pm

1. Problem Statement:

The goal of this programming project is to give students experience implementing and using a stack in a file processing application.

The HTML standard includes dozens of HTML tags that can be used to create web pages. Some tags control how a web page is formatted while others control user input or navigation. Most HTML tags come in pairs. We use <tag> … </tag> around a block of text to tell the browser what to do with the text between the tags. For example: "<h2>Title</h2>" will display the word "Title" in a large bold font. The HTML standard says that we are allowed to nest one set of HTML tags inside another set of HTML tags. For example: "<p>This is a <b>silly</b> example</p>" creates a paragraph with the word "silly" displayed in bold.

Unfortunately, when people create HTML pages by hand, they often forget to include all of the ending tags, or they nest the HTML tags incorrectly. This results in an invalid HTML file that may or may not be displayed correctly by a web browser, depending on how much "automatic syntax correction" the browser supports.
For example, the following page has several "missing end tag" errors:

```
<html>
<body bgcolor=white>
<center>
<table width=70% cellpadding=5 border=1 bgcolor=#990000>
<tr><td align=center>
<font size=+2 style=times color=white> CSCE 2004</font><br>
<font size=+5 style=times color=white> Programming
Foundations I</font><br>
</td></tr></table>
</center>
<h2>Class Website Links:</h2>
<p><li><a href="S17/index.html">Spring 2017</a>
<p><li><a href="F14/index.html">Fall 2014</a>
<p><li><a href="F13/index.html">Fall 2013</a>
<p><li><a href="F12/index.html">Fall 2012</a>
<p><li><a href="src/index.html">Source Code</a>
<p><li><a href="notes/index.html">Lecture Notes</a>
</body>
</html>
```

In this project, you will create a program that reads an HTML file, finds the HTML tags of interest, and checks to see if the HTML tags are all used correctly. Your first task is to implement and test a StringStack class. Your second task is to write a program that reads an HTML file and uses your new StringStack class to check if the HTML file is correctly following the HTML tag nesting rules. Your third task is to test your program with several simple web pages to see if they actually conform to the official HTML standard. Detailed instructions for this project are given below.

- <u>Implementing the StringStack class:</u> Start with the "stack.h" and "stack.cpp" files on the CSCE 2014 class website, and modify this code to create a stack of strings. You can do this by replacing "int" with "string" throughout the program, or by using a "typedef string DataType;" command before the class definition and replacing "int" with "DataType" throughout the program. There is NO need to use templates to make your stack fully generic.

- <u>Testing the StringStack class:</u> Start with the main program in the "stack.cpp" file and modify this code to test the basic operations of your StringStack class. You can compile this main program using "g++ -Wall –DSTACK_MAIN StringStack.cpp –o StringStack". By using the –D switch, you are essentially defining the STACK_MAIN variable, so the main program in "StringStack.cpp" will be compiled. If you leave off the –D switch, you will only compile the class methods.

- <u>Implementing the HTML checker</u>: In order to check to see if the HTML tags in a web page are used correctly, you must read the input file word-by-word or character-by-character to find all <tag> … </tag> pairs. When you find a start tag, you should push it on the StringStack. When you find an end tag, you should look at the top of the stack, and if the tags match, you should pop the tag off the stack. If the end tag does not match the start tag on the stack, you have found a tag nesting error. If the stack is not empty when you are finished processing the HTML file, this tells you that the user forgot an end tag somewhere. When your program finds an error with HTML tags, it should print out a message that tells the user what problem was detected.

- <u>Testing the HTML checker</u>: You should test your HTML checker using several simple web pages that were created manually. I suggest that you start with some pages on our class web site. You can copy these to your working directory by using "show page source" and cut/paste, or you can use the "wget" command on a linux machine to copy the file directly. Once you have the HTML file, run it through your HTML checker program to see if it is correct or not. If you find an error, save the input file and your program output for your project report.

2. Design:

You are starting with an existing stack implementation, so there is not much design needed to create the StringStack class. Your primary design task is to figure out how to read the HTML file to find start tags and end tags. If you make the assumption that all "<" characters mark the start of an HTML tag, and all ">" characters mark the end of an HTML tag, then you can use this information to help you when you are processing the input file.

Your next major task is to decide when to push tags on the stack. It turns out that there are a small number of HTML tags that do not require end tags (e.g. <br>, <hr>, <img>). To deal with this, you can create an array strings that contain the names of "unpaired" tags, and check this array before you push tags onto the stack. Alternatively, you can create an array of strings that contain the names of all of the "paired" tags that your program will check. You can start with the most common HTML tags and add to this list once your program has been debugged.

Your final task is to decide what your program should output when it is checking an HTML file. One option is to print all of the characters/words in the input file as they are read, and then print error messages when nesting errors are discovered. Another option is to print out just the HTML tags as they are read. You could also print out the contents of the HTML tag stack at various times if you want. You can use whatever output you think looks nice, and helps you debug your program.

3. Implementation:

This time, you are starting with "stack.h" and "stack.cpp" when implementing your StringStack class. You will have to create your HTML tag parser from scratch. It would probably be a good idea to work on reading text a character or word at a time to start, and then add code to detect start and end tags. When this is all working, you can add code to push and pop HTML tags in and out of your StringStack.

It is always a good idea to work incrementally, writing comments, adding code, compiling, debugging, a little bit at a time. Once you have the methods implemented, you can add calls to the methods to your main program to complete your project.

Remember to use good programming style when creating your program. Choose good names for variables and constants, use proper indenting for loops and conditionals, and type clear comments while you are writing the code. Also, be sure to save backup copies of your program somewhere safe. Otherwise, you may end up retyping your whole program if something goes wrong.

4. Testing:

Test your program to check that it operates correctly for all of the requirements listed above. Also check for the error handling capabilities of the code. Try your program with several input values, and save your testing output in text files for inclusion in your project report.

5. Documentation:

When you have completed your C++ program, write a short report using the project report template describing what the objectives were, what you did, and the status of the program. Does it work properly for all test cases? Are there any known problems? Save this report to be submitted electronically.

6. Midpoint Project Submission:

To encourage students to get an early start on their programming project, students are required to upload into Blackboard a partial solution to their programming project on the midpoint due date shown above. The program does not need to be complete, but it must compile and perform some of the tasks listed above. This midpoint solution is worth 10% of your final grade on the project.

7. Project Submission:

In this class, we will be using electronic project submission to make sure that all students hand their programming projects and labs on time, and to perform automatic plagiarism analysis of all programs that are submitted.

When you have completed the tasks above go to Blackboard to upload your documentation (a single docx or pdf file), and all of your C++ program files. Do NOT upload an executable version of your program.

The dates on your electronic submission will be used to verify that you met the due date above. All late projects will receive reduced credit:

- 10% off if less than 1 day late,
- 20% off if less than 2 days late,
- 30% off if less than 3 days late,
- no credit if more than 3 days late.

You will receive partial credit for all programs that compile even if they do not meet all program requirements, so handing projects in on time is highly recommended.

8. Academic Honesty Statement:

Students are expected to submit their own work on all programming projects, unless group projects have been explicitly assigned. Students are NOT allowed to distribute code to each other, or copy code from another individual or website. Students ARE allowed to use any materials on the class website, or in the textbook, or ask the instructor and/or GTAs for assistance.

This course will be using highly effective program comparison software to calculate the similarity of all programs to each other, and to homework assignments from previous semesters. Please do not be tempted to plagiarize from another student.

Violations of the policies above will be reported to the Provost's office and may result in a ZERO on the programming project, an F in the class, or suspension from the university, depending on the severity of the violation and any history of prior violations.