LabQuiz (20 points)  
 October 4 & 11, 2017

Laboratory Quiz (Revised)



**CS-2303, System Programming Concepts, A-term 2017**

### The Lab Quiz

This quiz is to assess your skills in using *Eclipse* and its debugger to debug *C* and *C++* programs. During this quiz, you need to demonstrate to one of the course staff that you are able to carry out the following routine tasks associated with debugging. Each task is worth one point. If you complete all 15, you get 5 bonus points. This brings the total value of the quiz to 20 points, the same as the weekly classroom quizzes during the term.

Note that you must be able to perform these steps without asking for help from the course staff during the quiz itself. In other words, you can ask for help while figuring out what to do, but at some point you have to:

* Say that you are starting the quiz.
* Start the steps from scratch.
* Not get any help after that.

#### Preparation

To prepare for the quiz, download the following two zip files from Canvas and expand them into a convenient folder or directory: LabQuizFiles.zip, LabQuizData.zip .

#### Quiz tasks

1. Create a new *Eclipse* workspace, and within that workspace, create a new *C++* Makefile Project with Existing Code,[[1]](#footnote-1) specifically from the files in LabQuizFiles.zip. Make sure that you can build it and clean it. Also, make a new folder called Data in the project. Copy the data files into this folder.
2. Set up the command line arguments so that the first argument is a file named Output.txt and is to be stored in the Data folder. The next three command line arguments should be the three text files that you copied into Data from LabQuizData.zip*.*
3. Debug the project to bring up the Debug Perspective. Before allowing the program to execute any lines, display the program arguments to main(). In particular, display argv as an array, and show all of the strings in argv. In addition, show that argv ends with a null pointer (i.e., after your last argument).
4. Run to line 35 of the function **main()**, where it tries to open the output file. Single-step over the next few lines to see if it was able to open the output file.
5. Run to line 50 of the function **ScanInputFile()**, where it attempts to open an input file. Single-step over the next few lines to see if the input files are opening correctly.

Run the program to completion and check the output file.

1. Set a conditional breakpoint in **int** **TreeNode::incr()** at line 45 (the **return** statement). Using the Breakpoint Properties dialog for this breakpoint, make this a conditional breakpoint to stop only when the **count** **==** **10**.

Re-run the program (with the Debug command) to this breakpoint. Display the variables and parameters of this function.

1. With the variables displayed, look inside the member **word**. This is an object of class **string**. You can see the word of input text that is stored in the **TreeNode** that is currently being incremented at this breakpoint. *Make a note of this word.* (Note: In previous versions of Eclipse you needed to expand the **string** variable and then expand its member **M\_dataplus**. The new version of Eclipse shows you the text without expanding.*)*
2. Edit the Breakpoint Properties of this breakpoint to set the condition to **count** **==** **5** and to ignore five instances of this breakpoint. Resume execution and note which word is in the **TreeNode** when execution pauses at this breakpoint. Also examine the Console window and make note of which input file is currently being scanned.
3. Look at the call stack. For each function in the call stack, click on it to examine its variables and to display the line in your code making that function call. Note how many recursive calls are open (i.e., functions that have been called by not yet returned).
4. Investigate the call to **BinaryTree::AddNode(TreeNode** **\*subtree,** **const** **string** **&word)**. Confirm that the word that it is trying to add is the same one that caused you to stop at this breakpoint.
5. Still at the same breakpoint, look at subtrees pointed to by the **left** and **right** members of the current **TreeNode**. Make a note of the words stored at the roots of each of these subtrees. Also make a note of the words stored in the four subtrees at the next level.
6. Disable all breakpoints and run to line 44 of Lab7.cpp (in function main()).
7. Explore the binary tree WordTree in the Variables tab. Make a note of the word at the root of the tree and the two words at the roots of the left and right subtrees, respectively.
8. Terminate the debugging session. Change the order of the input files in the Command Line arguments so that a different file is read first. Debug the program and run to line 44 of Lab7.cpp again. Note what words are at the root of WordTree and the roots of the left and right subtrees, respectively. Do they match your expectations from the text of the files?
9. Disable all breakpoints and run to the end of the program. Display the Output.txt file in a window to convince yourself that it worked.

That’s it.

### Getting Credit for this Lab

To get credit for this lab assignment, simply have an instructor, Teaching Assistant, or Senior Assistant watch over your shoulder while you execute these steps.

1. Make sure that you do this *exactly* as specified in Lab #2 (in particular, be sure you import the directory which contains the source files, not a directory above it). If you do not get it right, most of the other tasks in this quiz will not work as they should. [↑](#footnote-ref-1)