Computer Science II – Lab 5

Introduction

There is one major activity involved in this week's lab: debugging. It is built around the `text_count.cpp` program from Lecture 9. Please download a version of this from

```
http://www.cs.rpi.edu/academics/courses/fall04/cs2/lab05/text_count.cpp
```

Once you have downloaded this file, you need not access the network any further. Turn off all network connections.

The lab is quite short. Work on the assignment if you have extra time and discuss any problems that you are having with your TA.

Checkpoints

1. In the first checkpoint, you will change the function `count_word_occurences` to accept a `list` of strings instead of a `vector`. Make all the necessary changes in the code. Show the changes to your TA; compile your code and run it. There is a bug in the code, so don't worry if your output is messy.

2. In the second checkpoint, you will change the function `add_to_letter_counts` to accept a `list` of integers instead of `vector`; **do not implement this step!** List the issues that arise in a piece of paper and explain to your TA why such a change is a great/bad idea.

3. There is a very minor bug in the 4th function, `count_word_occurrences`. It is pretty easy to find, but this checkpoint is more concerned with how to find it rather than just finding it. Debugging with standard library containers is a bit tricky; debuggers don't always give you a clear view of what's happening inside the standard library classes. Therefore, we must use a combination of techniques: specially-written code to print the contents of containers at key points in the code together with the debugger that comes with your development environment (e.g. .net) to step through the code and watch the contents of simple variables.

   Therefore, start work on this checkpoint by adding a function to print a list of strings. Insert this function just before `count_word_occurrences` in your .cpp file. Insert calls to this function at the start of `count_word_occurrences` and after the sort. This will ensure that the list of strings is not corrupted before entering the function and will allow you to see the contents of the list after sorting. In addition, insert a `cout` statement just after the start of the `while` loop to print `*p` (the current string). Compile and link your program. You are ready to debug.

   Here's an outline of the steps to follow in debugging:

   (a) **Set a breakpoint:** In the source code, go to the first line of `count_word_occurrences` and click it. Select `Debug -> New Breakpoint`. Four options will be presented. You can set a breakpoint based on the program entering a function, at specific point in a file, at a memory address (not useful now), or when a specific data condition is met. Click `File` and you will notice that the current line number is displayed. Click `OK`. A breakpoint is now set. Your program will halt when execution reaches this location in the code (when run using the debugger).

(b) Look at the main menu and click `Debug -> Start` (F5). This will start your program under control of the debugger. A console display will immediately pop up. Provide some relatively simple input, but be sure to repeat some words. Type `^Z` for end-of-file. Your program will execute until reaching the breakpoint you've set.

(c) At this point, several windows will appear including the source code window. You should still be able to see the console where you typed the input, but this may be hidden. It is important to look back and forth between this and the .net display.

The default set of debugging windows includes an "Autos" and a "Call Stack" display. Autos displays local variables for your function. The Call Stack displays the current execution path (in terms of function calls). Double-click on different entries in the call stack and different code will be displayed. Be sure to get back to the code for `count_word_occurrences`. You can tell from the position of the yellow arrow.

(The Autos window does not provide much useful information about the standard library containers. They are much too complicated to be seen in this way. This is why we've added our own C++ code for output.)

You can add windows to the display. We will do so in just a minute.

(d) You can now step through your program one line at a time. Use F10 to step to the next line of code in the current function. This executes this line before stopping and redisplaying (it happens very quickly, though). This works even if the line involves a function call. If you want to see what happens inside the function call, typing F11 will put you into the called function's code. Try not to do this at calls to standard library functions. It can get messy. If you do, Shift-F11 will get you out.

Hit F10 several times and watch the console display. The output lines of code you wrote will appear so that you can see what's happening there.

(e) You can watch the status of a particular variable by clicking `Debug -> Windows -> Watch` and selecting one of the watch windows. Do this, and type in the variable name `count`, under the name column of that watch window. This will replace one of your displays. (You can get it back under `Debug -> Windows`.)

(f) Now, use F10 to step through the execution one line at a line. Look at the source code, the console, and the watch window. You should see the error pretty soon. When you have found it, show a TA how you found it using the debugger. This will complete this checkpoint and the lab.

Please note that this checkpoint has only given you a brief introduction to debugging. You will learn much more through extensive practice. If you have time at the end of the lab discuss any trouble you might have on this week's homework with your TA.