Lab: Head to Head Testing

Steven J. Zeil

October 16, 2013

Contents

There are many circumstances where you are working on a program but have access to a working (or nearly working) executable.

- In the real world, very few programming assignments are to write something new from scratch. Much more often, you will be adding new features to an existing program, or fixing bugs in an existing program, or trying to improve the performance of an existing program.
- Even if you are developing a whole new program, there may be an older one already in use that is being replaced because it is in a older programming language that no one in the company cares to maintain, or runs only on older hardware or operating systems that are due for replacement, etc.
- And in the academic world, you will often find that instructors[1] provide executable solutions to assignments so that you can see what the finished product of an assignment is supposed to look like.

Whenever you find yourself working on a system for which you already have a (nearly) working executable, you can take advantage of this by doing *head to head testing*, in which you

1. Run the old code on a set of inputs, capturing the outputs into a file.
2. Run your new code on the same set of inputs, capturing the outputs into a different file.
3. Compare the two output files to see what, if any, differences you can find between them.
    - Those differences might represent bugs in your new code.
    - Or they might be due to bugs in the old code that you have just successfully fixed.
    - Or they might represent the new behavior or functionality and that you were working to add ot the old program.

Whatever the reason, it's the *differences* in outputs that are particularly interesting. Spotting those differences can be quite tricky, however, if they are buried in a sea of nearly identical outputs or if the differences are "invisible" changes like adding blanks at the ends of lines or improperly indenting output.

Luckily, there are simple tools that you can use to find and highlight those differences.

[1] Like me!

1 Lab Instructions

1. Read the problem description below.

2. You will find files for this lab in this directory, or, if you are logged in to a CS Dept Unix machine, in ~cs250/Assignments/head2h

   Compile the program on Linux (use the command "`make`"). The executable should be named `triangle`. Rename it to `triangleA`.

3. Now introduce a simple bug of some kind into `triangle.cpp`. For example, swap the "else if" and output statements for the isosceles and equilateral cases in `report()`.

   Recompile the program and rename the executable to `triangleB`.

4. Run the input files provided through each version of the program. To start with, try running

   ```
   ./triangleA < test0.in
   ```

   just to be sure things are working.

   We saw the use of < to *redirect* input so we would not have to type everything manually in a much earlier lab.

   We can also use > to redirect output into a file that otherwise would have gone to our screen. So run

   ```
   ./triangleA < test0.in > test0A.out
   ./triangleA < test1.in > test1A.out
   ./triangleB < test0.in > test0B.out
   ./triangleB < test1.in > test1B.out
   ```

   You now have four output files. Inspect these with `more` or an editor.

5. Next we are going to explore some of the common tools for quickly and automatically comparing output files head-to-head to see if any differences exist and, if so, what those differences are.

   Unfortunately, although "diff" tools like these are widely available, they are not a standard part of Windows distributions. They are, however, available in just about any Unix or Linux system.

6. The most basic of the comparison tools is `diff`. Try running

```
diff test0A.out test0A.out
```

Now, since we are comparing a file against itself, obviously there are no differences to be reported. Then try

```
diff test0A.out test0B.out
diff test1A.out test1B.out
```

`diff` gives you a very quick report on the differences, if any, between two files.

The behavior of `diff` can be tweaked by various command options. For example,

```
diff −i test0A.out test0B.out
```

will ignore differences in upper and lower-case characters when comparing the two files. The option -w causes diff to ignore differences in white space (blanks and tabs).

7. `diff` is pretty straightforward, but the output isn't particularly pretty. Try this next:

```
sdiff test0A.out test0B.out | more
sdiff test1A.out test1B.out | more
```

(Hit 'q' to exit the "more" program.)

`sdiff` shows the two files "side by side". You may find it useful to make the window for your ssh session a bit wider when using this program. Lines that match exactly are shown side by side with no marker in between them. Lines that appear to have been changed have a '|'. Lines that appear to have been added or removed from one file but not the other are marked with a '<' or '>' pointing at the "extra" line.

8. If you have progressed in CS252 as far as the exercises on X, you might want to try the emacs in X assignment next. As part of that, you will be using a more interactive difference viewer in `emacs` that steps from one change to another, highlighting the differences in color.

9. Although this lab has focused on programs that you would use in a Unix or Linux environment, you can perform head-to-head testing in Windows environments as well. Even if your program writes to standard output, you can use output redirection in a Windows `cmd` window to capture the output into a file, just as you uses *input redirection* to a program in the earlier "Supplying Inputs to Programs" lab.

If you are seated as a CS Dept lab Windows PC, run the program **windiff**. (If you are not in a CS lab, use remote desktop to visit the CS Virtual PC lab and run **windiff** there.

From the `File` menu, select “Compare Files...”. When prompted for the files to compare, browse to the `test0A.out` and `test0B.out` files that you created in your Linux ssh session. (These should be available in your Z: drive.)

If all you get is a simple report saying that the files are “different”, try selecting “Expand” from the `View` menu.

10. Repeat the **windiff** process with the `test1A.out` and `test1B.out` files

    Windiff is not actually my preferred tool for this purpose. I recommend WinMerge (which comes in a portable version that can be installed on a USB flashdrive and executed on almost any Windows machine).

2 Problem Description: Triangle Diagnosis

This is a simple program that reads in three floating-point numbers at a time. These numbers represent the lengths of the three sides of a triangle. The program interprets these to classify the triangle and prints the name of the kind of triangle that it is. You may remember these terms from a geometry class long, long ago:

- *Equilateral* triangles have all 3 sides of equal length
- *Isoceles* triangles have all 2 sides of equal length and the third of a different length
- *Scalene* triangles have all 3 sides of different lengths

In addition, there are some less obvious cases:

- A triangle is called *degenerate* if it has no interior area. For example, if one or more of the angles of the triangle is zero degrees, of if one ore sides have zero length, the triangle degenerates into a line segment or, if all three sides are zero length, into a single point.
- Some combinations of lengths are *not a triangle* at all. For example, no triangle can have one size longer than the sum of the other two sides.

The program reads from the standard input, three numbers at a time, until end of input is reached. For each trio of numbers, it prints a triangle classification to the standard output.