CSE134A Section - Notes on Debugging

Greg Chun

October 11, 2002

1 Small Steps and Simple Test Cases

One key to minimizing debugging time is working in *small steps and testing often.* It is a natural tendency to want to add massive amounts of code all at once to feel like one is making faster progress. However, this strategy is more often counterproductive, as it greatly increases your "bug-hunting" time when bugs are inevitably introduced into your code. And often times when the bug is eventually discovered, it is something completely trivial and not at all what you thought it would be. Taking smaller steps lets you catch bugs as they arise. When something goes wrong, there are really only a few places the bug could be. In addition, if you find yourself completely baffled, it's much easier to rollback the changes you have made to get back to a working version of your code.

Another common tendency is to want to work with a single file, making any and all changes there, and testing just this one file. However, I have found it useful to create simple test cases to make sure any new functionality works perfectly in isolation before attempting to integrate it into your existing code. You may think you're just cluttering up your workspace with a bunch of small files, but these small, isolated files can actually help you later on. The addition of new features is almost always followed by some amount of "tweaking". The same files that allowed you to quickly test and integrate your new code is also there for when you want to tweak the features you have added. This falls directly in line with the philosophy of decomposing a problem down to its constituent parts so that less time is spent trying to figure out *where* the problem is, and more time can be spent solving it. Obviously, I'm not saying that one needs to do this for *every* piece of new code that is to be added. But it can be a useful strategy, especially in the case where the code that is being added is somewhat unfamiliar territory.

2 Useful Print Statements

Print statements are quite often your most valuable debugging tool. Here are some examples for discussion:

2.1 Printing out POST variables

```
<?
   print("<b>POST VARIABLES</b><br>\n");
   print("<table border=1>\n");
   foreach($_POST as $key => $value) {
      print("<tr><td>".$key."</td><td>".$value."</td></tr><br>\n");
   }
   print("</table>\n");
?>
```

2.2 Printing out imported POST variables

```
<?
   import_request_variables("gP","pref_");
   print("<b>POST VARIABLES</b><br>\n");
   print("<table border=1>\n");
   foreach($_POST as $key => $value) {
      $varname = "pref_".$key;
      print("<tr><td>".$varname."</td><td>".$$varname."</td></tr><br>\n");
   }
   print("</table>\n");
?>
```

2.3 A SQL Query

Example 1:

```
$result = mysql_query("SELECT family_name, first_name, grad_year from people
        WHERE first_name=$pref_first_name order by family_name");
```

Example 2:

```
$sql = "SELECT family_name, first_name, grad_year from people
        WHERE first_name=$pref_first_name order by family_name";
$result = mysql_query($sql);
```

3 Work with Multiple Windows

This may seem fairly obvious, but it is worth mentioning in any case. Whether you are working on a PC, Mac, or a UNIX box of some sort, the beauty of most terminal applications is the fact that you have the ability to open up multiple windows and work with all of them simultaneously. For instance, you could have one window dedicated to your vi or emacs process for editing your main php file, one window that contains just a UNIX shell, one window with an interactive mysql session, and of course, your browser window. Not only does this help you work faster, but it encourages the breaking down of your problems into smaller parts. If you suspect something is wrong with your SQL statement, it's very easy to simply go to your mysql session window to test it. If your php file cannot be found, or you seem to be getting access errors, you have a UNIX shell that you can use to poke around and look at permissions. Of course, you must be careful if you start moving files around in your UNIX shell, as you may actually move a file that you are in the process of editing in another window. Despite these potential "gotchas", I find this style of working much more productive.