CS 111 Spring 2014

Lab 1A Design Report

For our design project, we implemented the redirection operators  >>, <&, >&, <>, >| as suggested on the syllabus. The operators are implemented so that they behave exactly as they should in a normal bash shell.

To implement our design, we first added new data associated with a command by declaring several char fields in command-internals.h under struct command: Then, we made the types representing these commands in the tokenizer, and parsed these tokens into the command stream. In the tokenizer, first had to make a few changes to existing checking conditions. Since we already defined the > and < operators, we needed to edit their check conditions to check the cases of having a >, <, &, or | following right after. If any of those conditions are satisfied, we store the operators into the token stream, and set each token type to be one of the fields that we listed under struct command in command-internals.h. After we formed the token stream, we took parsed to tokens to make command structures. The overall implementation of the tokenizing and parsing are the same as how we originally implemented everything for 1A. Last, we made small changes in print-command.c to make sure that these new operators are recognized and that they would be printed out correctly.

Besides implementing new operators, we did not improve on any additional function. Our goal was just to make sure our implementations were 100% accurate and 100% functional. However, if we could improve on something, it would be modularity. Currently, our code is mainly divided into 2 sections: tokenizer and parser. An idea to improve modularity would be to group operators based on similarity. For example, the && and || could be grouped together, all the redirection operators could be grouped together, and the subshell could be in its own group. This improvement could make the code more organized and easier to debug. In addition, currently we have a lot of repetitive code, so improving modularity by using more helper functions could eliminate some of the repetition as well.

Our implementation works for all the test cases that we came up with, however, because there exists such a large amount of possible corner cases, we are not sure if our implementation would work as seamlessly as the actual bash shell. One obstacle that we encountered while implementing our design was that we needed to re-order the sequence of print statements in print-command.c. Our original design did not work with the new operators because they would not print in the correct sequence even though they were parsed correctly. However, it was easier than we expected to fix the issue; we simply re-ordered some print statements and that fixed everything. Another obstacle we encountered was that we had a bug where whitespace that occurs before a parentheses or whitespace that occurs after a parentheses was still being kept in the buffer, so it affected the resulting token stream, but this was also an easy fix. There were also other obstacles we encountered regarding invalid syntax. Despite this, we believe that our implementation most likely will fail at least some corner case that we did not think of, because the amount of corner test cases is simply too vast. We have provided a test-design-ok.sh and test-design-bad.sh to test our code.

We divided the work through the use of pair programming, with one person programming and one person overlooking. Because there wasn’t much code to be added or anything difficult that needed to be programmed, we felt that this was the best approach to complete the design problem.