CMPT 214: Programming Principles and Practice
Term 1 2016-17
Lab 11 – Stack frames and shell scripting

At the beginning of this lab, the lab instructor will give a brief presentation about symbolic links in the UNIX file system. This information will be helpful in answering question 2e. You can also read about symbolic links in pages 109 to 118 of the Sobell textbook.

Complete each of the tasks below. For all steps involving the use of LINUX/UNIX commands, place the command you used along with the resulting output (i.e. copy-and-paste from your terminal window) in a file called `lab11.txt`. However, do not include extraneous or superfluous commands or output; only include content relevant and essential to the specified task. Add your answers to questions 2d, 2e, 3e, and 6f. Then, with a text editor, add to `lab11.txt` identifying information to clearly distinguish which commands/output correspond to each task/question. When done, hand in `lab11.txt`, as well as your `move_columns.sh` file, through the moodle page for the lab. Use tuxworld for completing the lab exercise. This lab is out of a total of 20 marks; the number of marks allocated to each question is indicated below. Marks may be docked for extraneous, irrelevant, or superfluous content or for not following directions. Your submission is due at 11:55 p.m. on Thursday, December 1.

For lab, you will need various supplementary files. They are in file `Lab11Files.tar` associated with this lab. Download `Lab11Files.tar` and unpack it prior to beginning the questions or tasks below. You do not need to show a log of the download or unpacking.

1. (4 marks) Build an executable `factorial` program from `factorial.c`, making sure that debugging information is included by the compiler. Run `gdb` on `factorial`. Set appropriate breakpoints, and then run `factorial` within `gdb`. Continue executing `factorial` until execution gets to the point pictured by `factorial_stack.pdf`. At that point, perform a `backtrace` command to show all the stack frames. Then without advancing `factorial`, use appropriate `gdb` commands that do not involve "`backtrace full`" to output the values of variables `f` and `i` of function `main()` of `factorial`.

   At this point you can exit from `gdb`.

2. Perform steps (a), (b), and (c) in succession using `bash`, and then answer questions (d) and (e). There is one mark for the combination of steps (a), (b), (c).

   (a) Execute the command "`set -x`".

   (b) Execute the command "`/bin/ls -l` `` `which sh` `` `which bash` ``".

   (c) Execute the command "`set -`".

   (d) (1 mark) Explain the operation of the command in part 2b. For example, what commands are being executed, and in what order? Make sure to refer to the output generated because of the `-x` option being turned on. Write your answer in `lab11.txt`.

   (e) (1 mark) Based on the output in step 2b, explain the relationship between `sh` and `bash` on tuxworld. Write your answer in `lab11.txt`.

3. Perform steps (a), (b), (c), and (d) in succession, and then answer question (e). There is one mark for the combination of steps (a), (b), (c), and (d).

   (a) Execute the command "`rm -f foobar`" to ensure that there is no file called `foobar` in your current working directory.

   (b) Execute the command "`ls foobar`".

   (c) Execute the command "`echo $?`".

   (d) Execute the command "`echo $?`" again.

   (e) (1 mark) Explain the difference in output between parts 3c and 3d. Put your explanation in `lab11.txt`.

4. Consider the following pseudocode:

```
if the file called foobar exists then
     print "foobar does exist"
else
     print "foobar does not exist"
```

   On the command line (not in a shell script), implement the above pseudocode using a `bash if-else` statement, where the condition in the `if` statement involves ...

   (a) (1 mark) the `test` command.

   (b) (1 mark) the `[` command.

   (c) (1 mark) the `ls` command. Do not use `test` or `[`. Hint: redirect both the standard output and the standard error of `ls` to `/dev/null`.

   In `lab11.txt`, show the output of each of the above `if` statements for both possible cases—when file `foobar` exists, and when it does not exist.

5. (a) (3 marks) The supplementary file `substrates.txt` is a tab-delimited text file containing biochemical data. Using a combination of `cut(1)` and `paste(1)`, create a new file called `substrates2.txt` that has columns 4-7 of the original `substrates.txt` first, followed by columns 1-3, followed by columns 8-10. The procedure you use to do this may involve more than one command, and may involve the creation of temporary files. Like `substrates.txt`, `substrates2.txt` should be a tab-delimited text file. Finally, using `wc(1)`, confirm that the original file and the final file have the same number of lines, words, and characters, supporting a hypothesis that no information was lost during the column permutation process. If you used temporary files, make sure to remove them as part of your set of commands. The Sobell text has tutorial information on use of `cut(1)` and `paste(1)`; see pages 766 and 905, respectively.

   (b) (3 marks) Write an `sh` script called `move_columns.sh` that accomplishes the same thing as the series of commands you used in part 5a, but on a file whose name is specified on the command line. Your script should accept one argument—the name of the input file—and then output the columns, rearranged as above, to the standard output. Thus, you should be able to reproduce what you did in part 5a via the command "`./move_columns.sh substrates.txt > substrates2.txt`". Unlike step 5a, however, the script does not use `wc(1)` to confirm that the output has the same number of lines, words, and characters as the input (though you could do this outside of the script). Also, your script does not need to be able to read from the standard input. Hand in `move_columns.sh` as part of your lab submission. In `lab11.txt` show a log of invoking the script as described above.

   Note that your script must be executable by `sh` and must therefore start with "`#!/bin/sh`". A `bash` script is not allowed.

6. (2 marks) For this question, you will need the supplementary file `password.sh`. The most important aspect of `password.sh` is that it defines a `bash` function called `gen_pass` for generating random passwords. The details of how that function is implemented are not necessary for answering this question. Perform the following five steps in succession in `bash` and then answer the question in part 6f:

   (a) Invoke the command `gen_pass`. You should get a "`command not found`" error.

   (b) Invoke `password.sh` as a command via "`./password.sh`".

   (c) Again invoke the command `gen_pass`.

   (d) Execute the commands in `password.sh` using the `source` command.

   (e) Again invoke the `gen_pass` command.

   (f) State within `lab11.txt` which of the above steps (6(b) or 6(d)) allowed you to thereafter successfully use `gen_pass`.