CMPT 214: Programming Principles and Practice
Term 1 2016-17
Lab 4 - More UNIX and programming style

At the beginning of your lab period, watch your lab instructor perform a brief demonstration of the LINUX/UNIX command `printf`, which you will need to use for one of the questions in this lab exercise. Alternatively, read about this command in the Sobell text, pages 917 – 920.

Complete the tasks or problems below. For each task, copy-and-paste the contents of your terminal window (including the commands that you typed, and any output produced by the commands you gave) into a text file called `lab4.txt`. However, do not include extraneous or superfluous commands or output; only include content relevant and essential to the specified task. Unless otherwise specified, all commands should be run on tuxworld using the `bash` shell. Then, with a text editor, add to `lab4.txt` your solution to questions 4 and 5. Also with the text editor add text and identifying information to clearly distinguish which commands/output/code correspond to each task. Submit `lab4.txt` through moodle when done. This lab is out of a total of 16 marks, with each question (2a, 3b, 4, etc.) being worth one mark, with the exception of parts 1a and 1b, which are worth 2 marks each. Marks may be docked for extraneous, irrelevant, or superfluous content. The submission is due at 11:55 p.m. on Thursday, October 6. Note that the lab specification is three pages in length.

1. (a) The `<<<` operator in `bash` allows you to provide a string as standard input to a command. For example, this command will print the content of shell variable `VARIABLE` if it contains the letter "a".

   ```
   grep a <<< "$VARIABLE"
   ```

   That is, `grep(1)` is invoked with (search) pattern "`a`". Since no filename argument is provided to `grep(1)`, the command reads its input from the standard input. Because of the leading dollar sign ('`$`') before the variable name, the shell expands the variable `VARIABLE` to its value (a string). Finally, cause of "`<<<`", the shell arranges for this string to appear on the standard input to `grep(1)` using an invisible pipe.

   Write a command following the model above that will print the contents of the variable `POSTAL_CODE` only if it is a valid Canadian postal code (with uppercase letters and with a space in the middle). Demonstrate the operation of your command with cases where (the value of) `POSTAL_CODE` is a properly-formatted postal code, and where it is not. In each case, set the shell variable `POSTAL_CODE` to your test value prior to issuing the `grep` command. Remember to check cases like "`junkS7N 5C9garbage`".

   (b) Download the file `mailinfo.txt` that is available as an ancillary file for this lab. You do not need to show a log of downloading the file in `lab4.txt`.

   Devise a `grep` command that will output the lines from `mailinfo.txt` that start with an uppercase "I" or "O", end in "box", and contain at least one lowercase letter in between. There may be other

characters between the "I" or "O", and "box". For example, the following lines, among others, will be matches:

```
Inbox
Outbox
```

The following lines, among others, will not:

```
the Inbox
INBOX
Outbox1
InbOX
Ibox
I box
```

Demonstrate that your `grep` command works as specified. You will need two `grep` commands, one where it outputs the lines from `mailinfo.txt` that match your pattern, and a second `grep(1)` that outputs the lines that do not.

(c) Repeat question 1b, except output the *number* of lines matching the above pattern, rather than the lines themselves. Do this with a single, simple `grep` command; do not employ a pipeline.

2. Copy the file `/etc/passwd` to your current working directory (it could be a subdirectory of your home directory for completing lab 4). Call the destination file `passwd`. This file consists of several rows and columns, with rows separated by newlines and columns separated by colons. The first column represents a username. Use UNIX commands to perform the following tasks. For each task, your command should read as input the content of your local `passwd` file and output *only* the information requested, with no additional information. Each question must be answered using a single command, except as noted. The command may be a pipeline (a single command, involving a series of one or more pipes, "`|`"). More information on the format of a `passwd` file is available via the command "`man 5 passwd`".

   Note that for the purposes of working with file `passwd` (and to simplify the question) do not be concerned about the settings of environment variables `LC_ALL` or `LC_COLLATE`. Also use the default ordering criterion of `sort(1)` in completing these tasks.

   (a) Output the list of usernames in `passwd`. (You do not need to use pipes for this one). The output is to contain just the username field from the file.

   (b) Output the list of usernames in reverse sorted order.

   (c) Output the username that is "greatest" or "largest" (comes last when the usernames are placed in sorted order, or comes first when the usernames are place in reverse sorted order).

   (d) Output the first character of the "greatest" username.

   (e) The same as in 2d, except the character must be converted to uppercase.

   (f) Save the "greatest" username (c.f. question 2c) in a file called `max_username.txt`. Then use the `more` command (i.e. a second command) to output the content of `max_username.txt` to the standard output.

3. (a) Set the shell variable `COLUMN1_HEADING` to "Name" and `COLUMN2_HEADING` to "Student number". Then use the `printf` command along with the above variables to output the following to the terminal.

```
Name        Student number
John        123456789
```

   You can use a single `printf` command to print both lines, or complex command consisting of two consecutive `printf` commands (each producing one line of output) separated by '`;`'.

(b) Use the `printf` command to output *exactly* the following to the terminal. In doing so, do not use more escape characters than are absolutely necessary.

```
$COLUMN1_HEADING\t$COLUMN2_HEADING\nJohn\t123456789\n
```

4. Explain why it might be a good idea to always put braces around the body of an `if` statement in C/C++, even if the body contains just one line of code. Consider, for example, the following code fragment:

```
if (i==3)
    putchar( c );
```

Suppose a statement is added to the code as follows:

```
if (i==3)
    putchar( c );
    flag=false;
```

5. For each of the following pairs of code fragments, state which one you think has better coding style. In part (a) assume that there is significance to the choice of function name, as poor as it is.

(a) i.
```
cs(); /* clear screen to restart cycle */
```
ii.
```
cs(); /* restart cycle */
```

(b) i.
```
char *strncpy(char *s1, const char *s2, size_t n) {
    char *p;
    for (p=s1; *p && 0<n; --n)  /* Copy n chars */
        *p++ = *s2++;           /* from s2      */
    for (; 0<n; --n)            /* Write zero or */
        *p++ = 0;               /* more nulls  */
    return s1;
}
```
ii.
```
char *strncpy(char *s1, const char *s2, size_t n) {
    char *p;
    for (p=s1; *p && 0<n; --n)  /* Copy up to n   */
        *p++ = *s2++;           /* chars from s2  */
    for (; 0<n; --n)            /* Write zero or  */
        *p++ = 0;               /* more nulls     */
    return s1;
}
```

In this question, do not rewrite the code. Simply indicate which fragment of the pair demonstrates better programming style. Hence your answer in each case will be either "i" or "ii".