CMPT 214: Programming Principles and Practice
Term 1 2016-17
Lab 12 – awk and program linking

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 `lab12.txt`. However, do not include extraneous or superfluous commands or output; only include content relevant and essential to the specified task. Then, with a text editor, add to `lab12.txt` identifying information to clearly distinguish which commands/output correspond to each task/question. When done, hand in `lab12.txt`, as well as your files `population.awk` and `country_data_birthrate.txt`, through the moodle page for the lab. Use the tuxworld servers for completing the lab. This lab is out of a total of 18 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 8.

Note that this lab exercise description is three pages in length.

1. A file called `country_data.txt` is provided as supplementary data for this question. It consists of four columns: country name, population (in thousands), annual births (in thousands), and literacy rate (%). Before you start the tasks below, familiarize yourself with the contents of the file by examining its content with a program like `more(1)` or `less(1)`. You do not need to show a log of this in `lab12.txt`.

   (a) (2 marks) Like some data files you may encounter, `country_data.txt` is not formatted consistently—while most rows contain columns delimited by tabs, a few have one or more columns delimited by spaces. To verify this, use `cut(1)` to output only the third column (field), and redirect the output to `cutout.txt`. When you do this, do not change the default delimiter, which is a tab character. Then use `awk(1)` to output only the third column (field), redirecting the output to `awkout.txt`. Again, do not change the default delimiter, which in this case is one or more whitespace characters. Finally, use `diff(1)` to compare `cutout.txt` and `awkout.txt`. Submit a log of these steps. You do not need to submit `awkout.txt` or `cutout.txt`.

   (b) (3 marks) Write an `awk(1)` command to "clean up" `country_data.txt`. In other words, make it consistent that a single tab character, and only that, is used to delimit columns (fields) in the file. You will need to read in the file using `awk(1)`, then re-output all of the fields such that they are delimited only by single tabs. Place the output in a file called `country_data_cleaned.txt`. Your `awk(1)` script must be specified on the command line, rather than in a separate file. Then repeat question 1a with `country_data_cleaned.txt` to verify that the output of `cut(1)` and `awk(1)` are now identical when outputting the third column.

   Use `country_data_cleaned.txt` to answer questions 1c, 1d, and 1e.

   You do not need to submit `country_data_cleaned.txt`.

   (c) (2 marks) Write an `awk(1)` command that lists all the countries having a literacy rate less than 50%. Your command must output only the countries, not the other information in the table. Countries having an unknown literacy rate (represented by the "-" character) must not be included in your output. Your `awk(1)` code must be specified on the command line, rather than in a separate script.

   Remember to use `country_data_cleaned.txt` as your input file.

(d) (2 marks) Write an `awk(1)` command that prints the total population (in thousands) of all of the countries whose names begin with "A" and end with "a". Your `awk(1)` code must be placed in a script called `population.awk`, and then invoked using "`awk -f`" on the command line. Upload `population.awk` as part of your lab submission.

Note that this question does not ask for the population (total or otherwise) for each country whose name matches the pattern. Rather the `awk(1)` command is to determine the sum (the total) across all the countries whose names match the pattern.

Remember to use `country_data_cleaned.txt` as your input file.

Hint: the anchors '`^`' and '`$`', if used in a regular expression matching against a field, mean "beginning of the field" and "end of the field", respectively.

(e) (2 marks) Write an `awk(1)` command that reads `country_data_cleaned.txt`, and outputs the same information except with an additional column. This fifth column contains the country's birth rate (number of children per woman per year). Assume the population of each country is 50% male and 50% female. You may use the default output format for decimal numbers (7 decimal places). Like the other columns, the additional column must be delimited by a tab character. Your `awk(1)` code must be specified on the command line, rather than in a separate file, and the output of the command must be redirected to a file `country_data_birthrate.txt`. Upload your `country_data_birthrate.txt` file as part of your lab submission.

2. (2 marks) Use a pipeline involving `date(1)` and `awk(1)` to print out the current month and year in the following format:

```
Month: Nov
Year: 2011
```

Your `awk(1)` code must be specified on the command line, rather than in a separate script.

3. Consider the n-queens example from class where the source code for the program had been distributed among a number of source files. In that organization of the source code, `queens.cc` contained a short main program, functions related to finding a solution were in file `solver.cc`, and functions manipulating the game state were in `state.cc`. Appropriate interface information was placed in `solver.h` and `state.h`.

For the purposes of this lab exercise, the functions in `state.cc` and `solver.h` have been placed in a library libqueens on the tuxworld machines. The library can be found in `/student/214/lib`. The header files `solver.h` and `state.h` have been placed in `/student/214/include`.

Associated with this lab is a modified version of `queens.cc` from the aforementioned in-class example. The modification is that the lines

```
#include "state.h"
#include "solver.h"
```

have been replaced by

```
#include <state.h>
#include <solver.h>
```

That is, this version of `queens.cc` assumes that `solver.h` and `state.h` are "system include files". Download this `queens.cc` file, and use it as the source file in the remainder of this question. You do not need to show a log of the download in `lab12.txt`.

(a) (1 mark) Compile the downloaded `queens.cc` to `queens.o`, specifying that the compiler is to search `/student/214/include` for "system include files" (along with the other, default directories that it will search).

(b) (1 mark) Using `g++`, link and load `queens.o` to produce a dynamically linked executable file `queens`. Specify that the library `queens` for this lab is to be searched as part of the linking process. Also specify that in addition to the usual places that `ld` looks for libraries, it is also to search the directory `/student/214/lib`.

(c) (1 mark) Invoke your `queens` program from step 3b using the command "`./queens`". The program should result in an error

```
./queens: error while loading shared libraries: libqueens.so:
cannot open shared object file: No such file or directory
```

Fix the error by adding `/student/214/lib` to the setting of the environment variable `LD_LIBRARY_PATH` in your shell. Demonstrate that the `queens` program now works by finding a solution for $N = 4$.

(d) (2 marks) Using `file(1)` confirm that `queens` is a dynamically linked program. Then using `ldd(1)` examine the shared libraries that `queens` will make use of. Find in the output from `ldd(1)` the name of the file containing the `queens` shared library. Record that name in your `lab12.txt` file.

If you are up to a challenge and would like to take on some non-credit bonus lab questions, please contact the instructor.

And now you're done with the last lab in Cmpt214!