# CMPT 214: Programming Principles and Practice
# Term 1 2016-17
# Lab 5 - more grep and testing

---

At the beginning of your lab period, the lab instructor will review how to compile C and C++ programs on tuxworld with the `gcc` and `g++` commands. He will also demonstrate how to perform compilations using "`-Wall -Wextra`", and optionally "`-pedantic`".

---

Answer each of the questions below. For all questions involving the use of UNIX commands, place the command or pipeline you used along with the resulting output (i.e. copied-and-pasted from your terminal window) in a file called `lab5.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 `lab5.txt` your solution to questions 3(a), 3(b), and 4(f). Also with the text editor add text and identifying information to clearly distinguish which commands/output/code correspond to each task or question. Submit `lab5.txt` through moodle when done. Unless otherwise specified, all commands should be run on tuxworld using the `bash` shell. This lab is out of a total of 12 marks, with each question (1, 2, 3a, etc.) being worth one mark except for questions 2 and 4(a) which are each worth 2 marks. Marks may be docked for extraneous, irrelevant, or superfluous content. Your submission is due at 11:55 p.m. on Thursday, October 13. Note that the lab specification is three pages in length.

1. Use a UNIX/LINUX command to output the name of the tuxworld machine on which you are working. Then, on that machine, use a UNIX/LINUX pipeline to output all of the lines in `/etc/passwd` that contain the string "system", but do not contain the string "false".

2. Download the file `sentences.txt` which is available as an ancillary file for this lab. You do not need to show a log of downloading the file in `lab5.txt`.

   Use `egrep` to output all the lines in the file `sentences.txt` that have, somewhere on the line, the same word twice in a row. Your match should be case-sensitive. There must also be at least one non-word character between the two occurrences of the word. For example, the following lines, among others, would match:

   ```
   This is a sentence sentence.
   This is a a another sentence.
   this this is a sentence.
   ```

   The following lines, among others, would not match:

   ```
   This is a good sentence.
   This is a really good sentence.
   This this is a sentence.
   ```

Define a "word" to be an uninterrupted sequence of alphanumeric characters or underscore ('`_`'). Also, assume the C locale and ASCII character set encoding.

Make sure to test for matches as well as mismatches.

3. Consider the following code fragment, and then answer the questions below.

```
if (i > j) {
    printf("%d is greater than %d.\n", i, j);
} else {
    printf("%d is smaller than %d.\n", i, j);
}
```

   (a) Describe three tests that you would perform on this block of code to maximize the chance of discovering any bugs. In other words, what values for `i` and `j` would you try? You can assume `i` and `j` are integers.

   (b) Assume that you test with the test cases you specified in part (a). Consider what output would be produced by the code above for each test case. Describe one modification you would make to the code as a result of your testing.

4. Download the file `leap_year.cc` which is available as an ancillary file for this lab. You do not need to show a log of downloading the file in `lab5.txt`.

   A given year is a leap year if it is evenly divisible by 4 and is not evenly divisible by 100; however, years evenly divisible by 400 *are* leap years. You have been provided with a C++ file called `leap_year.cc` containing a (possibly buggy) C++ function called `isLeap`, which takes an integer (a year) as input, and decides whether it is a leap year according to the above rule. Function `isLeap` is *supposed* to work as follows:

   - If a given year $X$ is a leap year, `isLeap` returns the string "$X$ yes".
   - If a given year $X$ is not a leap year, `isLeap` returns the string "$X$ no".
   - If a given year $X$ is less than one, `isLeap` returns the string "$X$ error" (i.e. `isLeap` only works with years in the common era).

   where $X$ is replaced by the input (year) provided to it. In each result string, there is one space character between the year and the word "yes", "no", or "error". As well, the string is terminated by a newline ('`\n`') character. Years are integer values starting at 1.

   For instance, if `isLeap` were given the year 1996, it is supposed to return the string "1996 yes".

   Your job is to test `isLeap` by performing the following tasks:

   (a) In the `main` function of `leap_year.cc`, create a testing scaffold that will read in a sequence of years (integers), one per line, from the standard input until EOF (end-of-file). For each year, your `main` function must call `isLeap` and output the string it returns on the standard output. Finally, the `main` function returns with success (indicated by a return value of 0) when it gets to the end of the input file. Make your code for obtaining the input as simple as possible. For example, you can use a construction involving "`cin >> year`". Do not worry about deallocating the memory for the string returned by `isLeap`, although you can if you wish. *Do not modify any of the code in function `isLeap`.*

   (b) Create a sample input file called `leapin.txt` containing appropriate test cases for `isLeap`. Make sure to cover the kinds of test cases discussed in class. However, for simplicity, have all your test input be legal integer values.

(c) Create an expected output file called `expout.txt` that contains the exact output that is expected from your C++ program when `leapin.txt` is used as input.

(d) Compile and run your C++ program, redirecting input from `leapin.txt` and redirecting output to `leapout.txt`. If you use "`-Wall -Wextra`" you can ignore the warning(s) from the compiler complaining that variables `argc` and `argv` are not used.

(e) Use `diff` to compare the content of `leapout.txt` and `expout.txt`. Redirect the output from `diff` to a file called `leapdiff.txt`.

(f) Based on your output from step 4e, on what kinds of input does `isLeap` work correctly? On what kinds of input does it work incorrectly?

When done, submit `leapin.txt`, `expout.txt`, `leapout.txt`, `leapdiff.txt`, as well as your modified `leap_year.cc` (with the testing scaffold). In `lab5.txt`, include logs showing steps 4d and 4e, and your answers to the questions in part 4f.