CS4115 Week02 Lab Exercise

Lab Objective: We will use this first lab to set up a directory structure to use for the remainder of the semester, and to compare running times of pieces of code. Here's a quick summary of the tasks:

❶ Create a series of hierarchical directories (folders) to manage our work throughout the semester ahead

❷ Create your own copy of the two small programs that are provided and examine their behaviour in an editor

❸ Compile two small programs that are provided

❹ Compare their running times against each other

In Detail

❶ Note that there are a few alternatives given below for this first step so please read the entire instructions before acting. Over the semester we will have labs, programming assignments, etc. and it is a good idea to keep these in an organised way. My suggestion is that you create a subdirectory of your home directory called `cs4115` that will be the top-level point for all module-related material. The command to create a directory in Linux is `mkdir`, so you could do

```
mkdir ~/cs4115
```

which makes a subdirectory located in your home directory. Next you should make a subdirectory called `labs` in this for each week's work. This can be done with

```
mkdir ~/cs4115/labs
```

Finally, for this week's lab, Week02, you should create its own subdirectory with

```
mkdir ~/cs4115/labs/week02
```

An alternative to making each level of the hierarchy at a time is to tell mkdir to make the "parent" subdirectory if it doesn't exist. So the following command can take the place of all of the previous ones.

```
mkdir -p ~/cs4115/labs/week02
```

The `-p` is for "make parents if they don't already exist." Note that it is very similar to the command before it, but you had to do a lot of extra work in order to achieve that.

Yet another alternative is to bring up a file manager and use the “Create ...” menu option there.

❶ We are providing you with two little programs that have different running times. Firstly, you should copy these form the class account and then compile them.

To do the former you should give the command:[1]

```
cd ~/cs4115/labs/week02
cp ~cs4115/labs/week02/{lin,quad}.cc .
```

Firstly you change your *current working directory* with the `cd` command to `~/cs4115/labs/week02`. Then you, in effect, copy (`cp`) the two files named `lin` and `quad` that end in `.cc` to `.`, which is a shorthand for the *current working directory*, where we are now.

Watcher!!

Watch out! Please be aware of the distinction between `~/cs4115/labs/week02` and `~cs4115/labs/week02`. One is in your directory tree, while the other is in the class account directory tree. (On my keyboard the symbol “~” is a few to the right of the “L” key; it is called “tilde” and it is pronounced “til-dah”, as in “Waltzing Ma”.

At this point you should take a look at the programs you have copied over. I won’t try to direct you too much in what is a good source code editor but I like `emacs`. If you like you could also investigate learning about an IDE (Integrated Development Environment) which would allow you edit, compile, run, debug programs in one “integrated environment” but this is not necessary for now.

When you look at the programs in an editor the most important thing to watch for from an algorithm point of view is that `lin.cc` has one `for` loop, while `quad.cc` has two nested loops.

Each program is set up with a bit of magic so that we can supply a parameter (called `n` below) from the command line. This is done with the line of code:

```
int n = atoi(argv[1]);
```

In `lin.cc` the very same work is performed in each iteration of the `for` loop. In each iteration we call a function `count()` that counts the number of 1s in the file `/tmp/blah.txt`. Yes, it is stupid performing the same work repeatedly but the point of the exercise is to compare a one-loop program with a two-loop program. Please take a few minutes to look at how a file is opened in C++ .

The program `quad.cc` is, on the other hand, much simpler. We just have two nested

[1]You may be able to do this step using a graphical file manager but I don’t have details.

loops and at the guts of them is a divide statement and a simple assignment statement.

❸ Now you're ready to compile them. This can be done with the commands:

```
g++ lin.cc -o lin
```

and

```
g++ quad.cc -o quad
```

Just to make you work / sweat a little more I have deliberately put an error in `lin.cc` that prevents compilation. Please fix this error as the program will not work otherwise.

You will now have two executable files in your directory called `lin` and `quad`. These are abbreviations for programs that have, respectively, linear- and quadratic-running time.

❹ You can run these programs with the commands

```
./lin 1000
```

and

```
./quad 25001
```

where the `./` tells `bash`, the shell interpreter, that it will find this command in the current working directory.

Today's Tip

You can avoid always having to specify, `./`, the current working directory, by a quick edit of your `.bash_profile` in your home directory. In this file you will find a line that begins with "`PATH=`". What follows is a list of directories, each separated from the next by a `:`. Whenever you issue a command, this list of directories or locations is where the shell looks to try and find an executable program. So by putting just a simple `.` at the end of the line will now tell the shell interpreter to look in the current directory as a last resort for a command to execute. 1) Don't forget the `:` separator and, 2) This will work next time you log in; for the present session, issue the command `sh $HOME/.bash_profile`. From now on I will assume that you have adjusted your `PATH` so that the shell always searches in your current working directory when it is trying to execute a command (program).

Next time we will look at recording running times for the two programs and comparing them so as a warmer-upper for that play around with running the two programs for various values of `n`. Little can be learnt from very small values of `n` so I would use a base value of n=1000.

You can time a program's execution with the `time` command. The syntax in this case would be

```
time lin 2500
```

What you get back is three numbers that represent the no. of minutes and seconds that your program spent executing (*real*). This is broken down into the amount of time spent in your code (you're the *user*) and the amount of time spent in code owned by the *system*; examples of the latter would be computing sin, opening a file or, generally, running any code in a library not owned by you.

A function $f(x)$ is *linear* if, roughly speaking, $f(3x) = 3f(x)$ For the `lin` program verify that its name is appropriate by showing that if the size of `n` goes up by a factor of, say, 4, then the running time goes up by the same factor, roughly speaking.

What can you conclude about `quad`? Please keep in your mind at all times that running either program for small values will lead to unreliable answers. Sadly, as with so much of life, what we mean by *small* is not easy to say: it depends very much on context.

A stitch in time...

I have been harping on about my expectation that you make a strong commitment to the class by working outside of class hours, 4 hours being the minimum expectation. One thing that you could do to help yourself get ahead of the game would be for you to take seriously the job of learning C++ . You will have to do it sooner or later...
Liam Ryan (previously of this parish) has pointed me to the following site that is specially geared to people who have learnt Java and are learning C++ – that's you. Have a look at the following link:
`http://pages.cs.wisc.edu/~hasti/cs368/CppTutorial/`.

More next week.