15-122: Principles of Imperative Computation, Fall 2014
Lab 0: Starting Out with C0

Tom Cortina(tcortina@cs)

Monday, August 25, 2014

1 Navigating your account in Linux

Exercise 1. Open up a Terminal window to access the Linux command prompt. (Unlike typical graphical interfaces for operating systems, here you are entering commands directly to the OS and can change hundreds of options to give finer control of what you're doing.) On the Linux machines in the computer labs in Gates (rooms 5205, 5201, and 3000), you can access a Terminal window using the menu sequence:

```
Applications --> System Tools --> Terminal
```

Exercise 2. In the Gates computer labs, your terminal window will give you a prompt and you will be in your home directory in your Andrew account. Type

```
ls
```

(to list files) to see what files and directories are there. Back in the day, we used to call folders "directories". To move to another directory, use the cd command (for "change directory"). You should see a private directory in your listing. So change into that directory:

```
cd private
```

ACADEMIC INTEGRITY NOTE: You should store your program files and other class solutions inside the private directory (or a subdirectory inside this directory) since this directory is automatically set to prevent electronic access by other users. Remember that you should protect your work from being accessed by other students as part of the academic integrity policy for this course.

Exercise 3. Since you will write a number of programs for this course, it pays to make a subdirectory inside the private directory. Once you `cd` into the `private` directory, make a new directory named `15122` using the `mkdir` command:

```
mkdir 15122
```

Now go into this directory using `cd` again:

```
cd 15122
```

Once inside this directory, create another subdirectory called `lab0` for today's lab and then change into that directory:

```
mkdir lab0
cd lab0
```

Exercise 4. Verify you are in the `lab0` directory by entering the command `pwd` to get the *present working directory*. You should see something like this (with some user directory number and your andrew ID):

```
/afs/andrew.cmu.edu/usr_number/your_id/private/15122/lab0
```

Once you are in the `lab0` directory, copy a file containing a program into your directory:

```
cp /afs/andrew.cmu.edu/usr9/tcortina/public/15122f14/factorial.c0 .
```

WARNING: The final period is important and spacing is important too.

This command says to copy the file `factorial.c0` from the public directory `/afs/andrew.cmu.edu/usr9/tcortina/public/15122f14` to your current directory (specified by the shortcut of a single period). If you did this correctly, type `ls` and you should the file in the current listing in your directory.

2 Editing your program

You can use any editor you wish to write and edit your programs, but we highly recommend you try out `emacs` and `vim` since these editors can do much more than just help you edit your code (as you will see). For lab, **choose one of the two editors**, and following the instructions below. You can try out the other editor later.

Exercise 5. Open the `factorial.c0` file that you copied from the previous part of the lab:

EMACS: `emacs factorial.c0`
VIM: `vim factorial.c0`

(If the file doesn't exist, the editor will start with a new empty file.) You should see the editor start in the Terminal window, and you should see some program that looks like it computes factorial. The program is written in C0, the language we'll be using to start the semester.

Exercise 6. Edit the program and add your name and section letter at the appropriate locations. Use the instructions below for the editor you're using.

EMACS You can just start typing and editing without hitting special keys. You can use the arrow keys to navigate around the file to insert code. There are many shortcuts and built-in features to emacs but you don't need them right now. In the file, insert your name and your section letter in the appropriate comments in your program.

VIM This editor has two modes, an *insert mode* where you can insert text, and a *command mode* where you can enter commands. You start in command mode so you can't edit immediately. Use the arrow keys to move around the file. While in command mode, if you press `i`, the editor changes you to insert mode, allowing you to type text. In the file, insert your name and your section letter in the appropriate comments in your program. Press the Escape (ESC) key while in insert mode to return to command mode.

Exercise 7. Now save your changes and exit the editor. Use the appropriate instructions below.

EMACS Once you're ready to exit, press `Crtl-x` (the Control key and the x key at the same time) followed by `Ctrl-c`. Emacs will ask you whether you want to save your file (since you changed it) - press `y` for yes. (You could press `n` instead if you don't want to save your changes).

VIM *You can only save and/or exit while in command mode.* Save your work and exit the editor by entering the sequence `:wq` followed by the enter/return key. (You can exit without saving by entering the sequence `:q!` and followed by the enter/return key.)

For more information about using the editors: If you'd like to learn more about emacs editing commands, visiting this tutorial written by one of our teaching assistants: `http://www.andrew.cmu.edu/user/niveditc/pages/emacs.html`
To learn more about `vim`, you can try out `vimtutor` in your own time. Simply type `vimtutor` at the command prompt in the Terminal window to get started.

3 Compiling and running your program

When you run your program, it needs to be compiled first to check for syntax errors and, if none, to translate the code into a lower level (machine) version that can be executed by the computer. Before you do this for the first time, you need to make sure that we can access the compiler `c0` or interpreter `coin` from the command line.

Exercise 8. First, you need to find out what command shell you're using, since set up is different for each shell. At the command line prompt, enter:

```
echo $SHELL
```

If you see `csh`, then enter the following command, exactly as shown:

```
setenv PATH ${PATH}:/afs/andrew/course/15/122/bin
```

If you see `bash`, then enter the following command, exactly as shown:

```
export PATH=$PATH:/afs/andrew/course/15/122/bin
```

If you see another shell designation, ask your teaching assistant for assistance. **NOTE: There is a script you can run to set this up so it automatically happens each time you log in. Your teaching assistant can show you how to do this after you complete the lab or you can find instructions on our course website.**

Exercise 9. Compile your code using the `cc0` compiler as discussed below. (Be sure you're still in the `lab0` directory.)

```
cc0 -d factorial.c0
```

This runs the compiler with debug mode on (`-d`). Debug mode includes all of the annotations starting with `//@` as part of your code for testing. You will learn how these work in class. If there are no syntax errors, the command line prompt will be displayed without any other messages. Type `ls` and you will see a new file in your directory named `a.out` which is the executable version of your program. (If you have syntax errors during compilation, go back into the file with an editor and correct them.)

Now run the program by typing in the following command:

```
./a.out
```

The first dot says to look in the current directory (recall this shortcut from the `cp` command) and run the `a.out` executable file. This will cause the `main` function in your program to launch, which prints the values of 0! through 9! in the terminal window, one per line.

Exercise 10. Another way to run the program is *interpreted mode* where the instructions are checked, translated and run in one step. This is a good way to interact with your program in real time to test it carefully. Execute the following command to run the interpreter on the factorial program:

```
coin -d factorial.c0
```

You should see the interpreter launch, waiting for a command:

```
C0 interpreter (coin) 0.3.2 'Nickel' (r364, Tue Nov 19 22:45:16 EST 2013)
Type '#help' for help or '#quit' to exit.
-->
```

Enter the following command in `coin` to run the factorial function to find 10!:

```
factorial(10);
```

Enter the following command in `coin` to run the factorial function to find 17!:

```
factorial(17);
```

(Does anything strike you as odd here? We'll find out why soon.)

Enter the following command in `coin` to run the factorial function to find −1!:

```
factorial(-1);
```

In this case, you should see an annotation failure. This is because in our code, our factorial function starts with the requirement:

```
//@requires n >= 0;
```

Since we called this function with a value for n that does not satisfy this requirement, we get an annotation failure since it doesn't make sense to run this function with `n` equal to -1.

To exit out of the interpreter, enter. `#quit` at the prompt.

Exercise 11. Start the interpreter again, this time without the `-d` flag:

```
coin factorial.c0
```

Now try to compute the factorial of -1 as you did before. This time, you get no error message. Why do think this is? Exit out of the interpreter again.

Exercise 12. As you type code in C0, you will likely get a lot of syntax errors, so you need to read them carefully to figure out what the compiler is telling you. In this final exercise, use the text editor you chose to edit the `factorial.c0` file and change one instruction to introduce a syntax error. Save and exit from the editor and then use the compiler or interpreter to generate a syntax error. Read the error and see if it makes sense based on the edit you did. Does the error include a line number? If so, was the error on that line number, or before that line number? Go back into the editor and fix the program and make sure it compiles and runs successfully.

FUN ACTIVITY If someone sitting next to you is also done, edit your factorial function so it generates a syntax error, and see if they can figure out what you did!

4 Handin

There is no hand in required for Lab 0. Your teaching assistant will give you insructions on what to do next.