15-122: Principles of Imperative Computation

Lab Week 1, Monday — Tom Cortina, Rob Simmons

Navigating your account in Linux

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.)

(1.a) Open up a Terminal window to access the Linux command prompt.

On the Linux machines in the computer labs, you can access a Terminal window using the menu sequence: Applications $\longrightarrow$ System Tools $\longrightarrow$ Terminal. Your terminal window will give you a prompt and you will be in your home directory in your Andrew account.

(1.b) At the prompt, list the files in your home directory by typing

```
% ls
```

We often use the % or $ character at the beginning of a line to indicate that it's something you're supposed to enter in at the prompt. Don't actually type it in! Just type "`ls`" and press Enter, don't type "`% ls`" and then press Enter.

(1.c) You should see the directory `private` as one of the files in your home directory. Move to this directory using the `cd` command (`cd` for "change 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.

(1.d) 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
```

(1.e) Use `cp` to copy the directory `lab-setup` from our public folder to your `122` folder.

```
% cp -R /afs/andrew/course/15/122/misc/lab-setup .
% cd lab-setup
```

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

This copies the folder `lab-setup` from the public directory `/afs/andrew/course/15/122/misc/` to your current directory (specified by the shortcut of a single period). Usually `cp` just copies files; the `-R` tells it to copy the directory and its contents.

(1.f) Verify you are in the `lab0` directory by entering the command `pwd` to get the present working directory and by entering the command `ls` to see the files in that directory. You should see something like this with your andrew id instead of `your_id`:

```
% pwd
/afs/andrew.cmu.edu/usr_number/your_id/private/15122/lab-setup
% ls
factorial.c0
```

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.

(2.a) Open the `factorial.c0` file that you copied from the previous part of the lab:

EMACS:

```
% emacs factorial.c0
```

VIM:

```
% vim factorial.c0
```

If a 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.

(2.b) 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, *insert mode* where you can insert text, and *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.

(2.c) Save your changes and exit the editor.

EMACS: Once you're ready to exit, press Ctrl-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 pressing Enter. (You can exit without saving by entering the sequence "`:q!`" followed by Enter.)

If you'd like to learn more about emacs editing commands in your own time, try this tutorial written by one of our teaching assistants: `http://www.andrew.cmu.edu/~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.

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 you can access the compiler cc0 or interpreter coin from the command line.

(3.a) Find out what command shell you're using, since set up is different for each shell.

```
% echo $SHELL
```

What you do next depends on what you see when you ran that command.

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

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

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

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

If you see another shell designation, ask your teaching assistant for help.

(3.b) Compile your code using the `cc0` compiler:

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

This runs the compiler with debug mode on (`-d`).

Debug mode checks all the code annotations starting `//@` for testing. You will learn how these work in class.

If there are no syntax errors, the `cc0` compiler returns to the command prompt without saying anything else. Running `ls` will show you 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.

(3.c) Run the program:

```
% ./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.

That shows us how to *compile* **(3.b)** and *run* **(3.c)** our programs. 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. The name of the C0 interpreter is `coin`.

(3.d) Run your program in the Coin interpreter, starting it with `coin -d factorial.c0` and entering in the five C0 statements as shown below.

```
% coin -d factorial.c0
C0 interpreter (coin)
Type ‘#help’ for help or ‘#quit’ to exit.
--> factorial(2);
--> factorial(3);
--> factorial(4);
--> factorial(10);
--> factorial(17);
```

Some of the outputs Coin gives should strike you as odd. We'll learn about what's going on there a week from tomorrow. Be patient for now ☺.

(3.e) Factorial $n!$ is only defined on nonnegative numbers. Try to compute a non-existent factorial:

```
--> 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 $n = -1$.

(3.f) Exit the interpreter:

```
--> #quit
```

(3.g) Start the interpreter again, this time without the `-d` flag:

```
% coin factorial.c0
C0 interpreter (coin)
Type ‘#help’ for help or ‘#quit’ to exit.
--> factorial(-1);
```

Making C0 work the next time you do something

Step **(3.a)** only set up `cc0` and `coin` to work this one login, and you don't want to have to repeat that every time you log in.

(4.a) Go to `http://c0.typesafety.net/tutorial/C0-at-CMU.html` and do the **second** step, *Setting up your environment*.

(4.b) Test that this works by **logging out of your computer**, logging back in, opening a terminal, and typing

```
% coin -d
```

If the command above correctly starts the interpreter, then you're set up! **Don't forget to log out again before you leave.**

(4.c) If you have your laptop, you should work through the first step of C0 at CMU, *Connecting through SSH*, on your laptop.