CS 3110 Fall 2014
Problem Set 0
Version 1 (last modified August 25, 2014)

You do not need to submit anything for this assignment. In that sense, it is optional. But we *strongly encourage* you to complete it now.

Objectives: This assignment will help you install an OCaml development environment. You will become familiar with tools used in the course: a virtual machine, the Linux shell, and the `cs3110` tool.

Getting help: When you need help, there are many resources available to you. The CS 3110 Piazza site is a great place to ask questions. Course staff and other students are very active and will typically respond within a couple hours. Consulting hours are a great place to ask about anything in this assignment and future assignments, as well as other questions you may have about OCaml or setting up your environment.

Exercise 1.

We have pre-loaded a Linux virtual machine image with all of the software you will need in this course as well as a variety of popular text editors and development environments. Download, install, and launch the virtual machine:

(a) Download and install Virtual Box for your operating system:

`https://www.virtualbox.org/wiki/Downloads`

(b) Download the 3110 virtual machine image:

`https://cornell.box.com/s/ciqym8cjdlds8hvayg4t`

It's a big file and will take some time to download.

(c) Import the virtual machine:

- Run VirtualBox.
- Select File → Import Appliance.
- Select "Open Appliance", and choose the `.ova` file you just downloaded. Then click continue and then click "import". This step might take some time.

(d) Run the virtual machine:

- Select cs3110-VM from the list.
- Click "Start".

(e) Use the controls found in the lower left-hand corner of the screen to perform these tasks:

- Launch a web browser.
- Open a terminal window.

Exercise 2.

You can interact with the OCaml interpreter in a terminal window using a *top-level* or *REPL* (read-eval-print loop) called `utop`. It repeatedly reads OCaml expressions, evaluates them, and prints the resulting value (and the type of that value).

Start `utop` (by opening a terminal window, typing “utop”, and pressing return) and enter each of the following expressions. You must end each expression with a double semi-colon `;;` followed by the return key to tell `utop` that you are finished with entry. Determine the type and value of each expression as evaluated by `utop`.

(a) `7 * (1 + 2 + 3)`

(b) `"cs " ^ string_of_int (310 * 10 + 10)`

(c) `let f x = x ^ "doz" in f "zar"`

To exit `utop`, enter `C-d`, that is, Control-d.

Exercise 3.

Many tasks can be most efficiently performed in a terminal window using an interface known as the *shell* or *command line*. With the shell, the user types in commands, such as

```
echo "Hello World"
```

and the shell issues those commands to the operating system, which executes them. There are many shell tutorials available on the web, such as:

`http://linuxcommand.org/learning_the_shell.php`

The lecture notes from CS 2043 Unix Tools and Scripting contain a wealth of information:

`http://www.cs.cornell.edu/courses/cs2043/2014sp/`

Complete the following tasks using the shell on the virtual machine:

(a) In your home directory, create a directory called `cs3110` (hint: `mkdir`).

(b) Now within the `cs3110` directory, create a directory called `ps0` (hint: `cd`).

(c) Use `echo` to create a file `cs3110/ps0/hw.txt` containing the string “hello world” (hint: I/O redirection).

(d) Remove the `hw.txt` file (hint: `rm`).

(e) Using `history` and `grep`, create a file `mkdirs.txt` in the `ps0` directory that contains all of the `mkdir` commands you have executed so far. Use `cat` to see the contents of `mkdirs.txt`.

(f) Use `touch` to create a file `output.nosubmit` in the `ps0` directory.

(g) Use `zip` to create a file `ps0.zip` in the `cs3110` directory. The zip should include all the contents of the `ps0` directory except `output.nosubmit`. Check the directory structure and contents of `ps0.zip` using `zipinfo`. The zip should contain exactly two files: a directory named `ps0` and a regular file named `ps0/mkdirs.txt`.

Exercise 4.

In this course, we will use a real-world tool that facilitates collaboration called `git`. It is a *distributed version control system*, and use of it will make working with a partner an easier and more rewarding experience.

(a) First you need to create a student account on GitHub, where many developers host their projects. Follow the instructions on the main page of GitHub to create an account:

`http://github.com/`

When signing up, use your `@cornell.edu` email, so that you are eligible for free private repositories. After GitHub recognizes your academic status and grants you the ability to create free private repositories, you can change to your personal email. (Although we suggest GitHub as a hosting service, you may use other services if they enable private repositories. Bitbucket is a popular alternative.)

> *Private repositories are of the utmost importance. A public* `git` *repository would share your code with the entire world, including your classmates, thus violating the course policy on academic integrity.* ***Therefore we require that you keep all your CS 3110 related code in private repositories.***

(b) Follow the GitHub tutorial to familiarize yourself with `git`:

`https://help.github.com/articles/set-up-git`

(c) To continue this problem set, you need to download some code using `git`. Navigate to an appropriate directory of your choice, then issue the following command:

`git clone https://github.com/cs3110/tutorial.git`

After doing this, use `ls` to see that you now have a new subdirectory called `tutorial` containing a file named `hello.ml`.

Navigate to the directory containing the file `hello.ml` from the release code, start `utop` again, and execute the following command:

```
#use "hello.ml";;
```

This causes `utop` to interprets all the OCaml code inside `hello.ml`. You should see the following output:

```
OCaml is awesome!- : unit = ()
```

Exercise 5.

The course staff has written a tool called `cs3110` that simplifies the task of compiling, running, and testing OCaml programs. You will use this program for all problem sets in this course. The staff will also use a variant of it for grading.

(a) The `cs3110` tool provides many commands, including the following:

```
cs3110 compile <file>   Compiles into a bytecode executable↘
   →. Relies on ocamlbuild.
cs3110 run <file>       Runs a compiled executable.
cs3110 test <file>      Checks unit tests in a compiled ↘
   →executable.
cs3110 clean            Removes all of the files generated ↘
   →during compilation.
cs3110 inspiration      Gives a healthy dose of inspiration↘
   → to the weary CS 3110 champion
cs3110 help             Explain a given subcommand
```

Type `cs3110 inspiration` and verify that you feel inspired.

(b) Navigate to the `tutorial/grep3110` directory, which contains an OCaml implementation of a simple search command similar to the built-in `grep` utility. The code for this program is divided between source files `file_utils.ml`, `regex_utils.ml`, and `grep3110.ml`, and unit test files `file_utils_test.ml` and `regex_utils_test.ml`. To start, compile the main program:

`cs3110 compile -l str grep3110.ml`

We use the `-l str` flag because `grep3110.ml` has a dependency on the OCaml `Str` library of OCaml, whose documentation you can view here:

`http://caml.inria.fr/pub/docs/manual-ocaml/libref/Str.html` .

Verify that you have a new directory `_build` and a binary executable `grep3110.d.byte` in that directory.

(c) The directory `tutorial/sample_files` contains a couple small text files. Use `grep3110` to search for any lines containing the string “fox” in `test2.txt`:

```
cs3110 run grep3110 "fox" ../sample_files/test2.txt
```

Verify that the output produced by running this command is the following:

```
Line 4: fox
```

(d) Let’s run some unit tests. First, compile the unit tests:

```
cs3110 compile file_utils_test.ml
cs3110 compile -l str regex_utils_test.ml
```

then run them:

```
cs3110 test file_utils_test
cs3110 test regex_utils_test
```

You should see no output when you run the `cs3110 test` commands, meaning that the unit tests pass.

(e) Let's add a new unit test. To make it interesting, we'll write a test that fails. Add the following lines (how? see the next exercise) at the end of `regex_utils_test.ml`:

```
TEST_UNIT "bogus" =
  let p = regex_of_string "Haskell" in
  let s = "OCaml" in
  assert_true (matches p s)
```

This unit test checks whether the string `"Haskell"` occurs in the string `"OCaml"`, which is obviously false. Recompile and test:

```
cs3110 compile -l str regex_utils_test.ml
cs3110 test regex_utils_test

File "regex_utils_test.ml", line 56, characters 0-113: ↘
   →bogus
threw Assertions.Assert_true("false is not true").
  Called from file "lib/runtime.ml", line 227, characters ↘
     →71-75
  Called from file "lib/runtime.ml", line 189, characters ↘
     →39-45
```

This time, `cs3110 test` produces output indicating that a unit test failed.

Exercise 6.

Begin to familiarize yourself with at least one of the following text editors on the virtual machine. All can be launched from the shell.

Emacs (`emacs`) Emacs supports extensive integration with a large number of languages and environments. Emacs is powerful and extensible, and has excellent extensions for OCaml, but it has a steep learning curve. There is a custom 3110 tutorial for Emacs:

`http://www.cs.cornell.edu/courses/cs3110/2014fa/courseware/emacs_guide.pdf`

Vim (`vim` or `gvim`) Vim is designed for very rapid text navigation and editing. Vim is also powerful but has a steep learning curve. It has an accompanying tutorial that can be launched with the `vimtutor` command.

Sublime (`subl`) A simple text editor with a gentle learning curve.