Data Structures and Functional Programming	**Problem Set 0**
CS 3110, Spring 2014	**Thursday, January 30**
Version: 1	**Last Modified: January 24, 2014**

Objectives

This **optional** assignment will help you install an OCaml development environment, and familiarize you with the basic tools that will be used in the course: a virtual machine, the Linux command-line interface, and the CS3110 tool. There is nothing to submit—simply follow the instructions below.

3110 Virtual Machine

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. The next few exercises will take you through the steps needed to install the virtual machine and familiarize yourself with it.

Exercise 1:

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/acqwpvnidu5yq1osd8lb`

(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". Note: this step may take some time.

(d) Run the virtual machine:

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

Exercise 2:

Familiarize yourself with the virtual machine by performing each of the following tasks. These controls can be found in the lower left-hand corner of the screen:

(a) Launch a web browser

(b) Open a terminal window

(c) Switch to the second desktop, and launch a terminal window

Shell Basics

In Linux, many tasks can be most efficiently performed using a command-line interface known as the **shell**. With the shell, the user types in commands, such as

```
echo "Hello World"
```

and the interpreter issues those commands to the operating system, which executes them. A special program `man` prints the **manual** for any command. The lecture notes from CS 2043 Unix Tools and Scripting contain a wealth of information:

`http://www.cs.cornell.edu/courses/cs2043/2012sp/`

In addition, there are many tutorials available on the web such as:

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

Exercise 3:

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

(a) In your home directory, create a directory called `projects` (hint: use the `mkdir` command).

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

(c) Use the `echo` command to create a file `projects/ps0/hw.txt` containing the string "hello world" (hint: use I/O redirection).

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

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

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

(g) Use the `zip` command to create a file `ps0.zip` in the `projects` directory. Do not include the `output.nosubmit` file. Check that this zip file has the right directory structure and contents using the `zipinfo` command.

Text editor

The following text editors are available on the virtual machine:

Emacs (run from the command-line using `emacs` or `xemacs`) Emacs supports extensive integration with a large number of languages and environments. Emacs is powerful and extensible, but it has a steep learning curve.

Vim (run from the command-line using `vim` or `gvim`) Vim is designed for very rapid text navigation and editing. Vim is also powerful but has a steep learning curve.

Sublime (run from the command-line using `subl`) A simple text editor with a gentle learning curve.

Exercise 4:

Familiarize yourself with at least one of the text editors on the virtual machine.

A First Taste of OCaml

OCaml provides two ways to execute programs written in the language: by compiling or interpreting them. You can interact with the interpreter using the OCaml "toplevel" `utop`. It takes a sequence of OCaml expressions, evaluates them, and prints the final result (and its type).

Exercise 5:

Using `utop`, determine the values and types for each of the following expressions:

(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`, press `^D` (control-D).

Exercise 6:

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

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

This interprets the OCaml code within it, and prints the final value. You should see the following output:

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

CS 3110 Tool

We have written a program `cs3110` that simplifies the task of compling, running, and testing OCaml programs. You will use this program for all problem sets in this course. We will also use a variant of it for grading.

Exercise 7:

The `cs3110` program provides the following commands:

```
Usage: cs3110 COMMMAND [args]

  cs3110 compile <file>   Compile file.ml.
  cs3110 run <file>       Run the program file.ml.
  cs3110 test <file>      Run the tests in file.ml.
  cs3110 clean            Removes files created by 'cs3110 compile'.
  cs3110 help             Displays this message.
```

Type `cs3110 help` and verify that you see the usage message above.

Exercise 8:

Download the `ps0.zip` from CMS. Move this file into your home directory (or anywhere you like) and unzip it by typing `unzip ps0.zip`. This will create a directory `ps0` populated with several subdirectories and folders including this writeup, and starter code.

Exercise 9:

Navigate to the `ps0/release/grep3110` directory, which contains an OCaml implementation of a simple search command similar to the built-in `grep` utility. You can use it to search for a string in a file. 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, let's compile the main program itself:

```
cs3110 compile grep3110.ml
```

Using `ls` verify that this creates a new directory `_build` and a binary executable `grep3110.d.byte` within that directory.

Exercise 10:

Now let's run the program on a file containing some text. The directory `sample_files` contains several small text files. For example, the file `test2.txt` contains:

```
the
quick
brown
fox
jumped
over
the
lazy
dog
```

Use `grep3110` to search for any lines containing the string "fox":

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

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

```
Line 4: fox
```

Exercise 11:

Next, let's run some unit tests. First, compile the unit tests,

```
cs3110 compile file_utils_test.ml
cs3110 compile regex_utils_test.ml
```

and then run them:

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

If all goes well, you should see no output when you run the `test` commands.

Exercise 12:

Now let's add another unit test. To make it interesting, we'll write a test that fails. Add the following lines 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. More generally, the syntax `TEST_UNIT str = exp` where `str` is any string and `exp` is an expression will succeed if `exp` evaluates to `()` and throw an exceptions reporting that the test named `str` failed:

```
File "regex_utils_test.ml", line 56, characters 0-110: 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
```

We will use unit tests often in this course.

Getting Help

This assignment is optional and there is nothing to submit. However, if 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.