CS 4: Fundamentals Of Computer Programming, Winter 2017

Assignment 0: Getting set up

Due: *Friday, January 20, 02:00:00*

Coverage

This assignment is independent of the textbooks.

Purpose of this assignment

The purpose of this assignment is to make sure that everything you need in order to do the assignments for this class is set up correctly. Note that this is a real assignment and you will get real marks for completing it successfully. It's worth a maximum of 2 marks.

NOTE: This assignment will be due on the same day as assignment 1. This may seem unfair or rushed, but it's necessary so we can get the full set of 7 homework assignments in by the end of the term, along with the midterm and final. This assignment should take very little time other than the time needed to download the virtual machine, and it's basically just two easy marks.

Grading

Normally, assignments in CS 4 are given integer grades from 0 to 3 (the last one may include fractional parts), as described on the web site. This particular assignment is intended to get you set up as quickly as possible, so the way we will grade it is different from all subsequent assignments. We will give you 2 points if you answer all questions correctly and submit your homework on time, 1 points if you answer all questions correctly and are late but no more than 1 week late; 0.5 mark if you answer all questions correctly and are between 1 and 2 weeks late, and 0 marks after that. The point is to make sure you do this assignment promptly and correctly, because the rest of the course will depend on it. You are welcome to ask TAs for help on any part of this assignment if you are having trouble following the instructions.

Prerequisites

We will be assuming henceforth that you have taken and passed CS 1 (or placed out of it). We will also assume that you have a rudimentary knowledge of Linux terminal commands (at roughly the level required for CS 1). If not, please read a Linux tutorial, because we will be using the Linux terminal a lot in this course.

Things you should have already done by now

In order to be able to submit this assignment (and any of the other assignments in this course), several things must have been done by now. If you've been to the lectures and/or read the lecture slides and/or read the information on the course website, you should know this, but every year, a number of students apparently don't do those things, so we are repeating this here for clarity, because we know that if you read nothing else, you will read the assignments. **PLEASE DO NOT SKIP THIS SECTION!**

1. You should have enrolled in the course Moodle page. Apparently you did this, or you wouldn't be able to read this document. Well done! :-)

2. You should have a CMS cluster computer account. Without this, you won't be able to submit your homework, whether or not you intend to use the CMS cluster computers. If you have an account from a previous term, you should have checked it to make sure that it still works (they get deactivated if you don't use them for an extended period) and if it doesn't work, you should have gone to see the system administrators in Annenberg room 112 between 9 and 5 to reset it and/or emailed them at help@cms.caltech.edu. (Going in person is better because they can fix it instantly.) If you have never had a CMS cluster account, you should have applied for one here and changed your password promptly once you received email notice that the account was set up.

3. You should have filled in the "CS 4 signup sheet" on the course website. Without this, we can't add you to the course csman page, which means that you will not be able to submit assignments. **THIS IS INCREDIBLY IMPORTANT!** Every year, a bunch of students forget to do this and then try to email their assignments to the TAs and/or the course instructor. **Emailed assignments do not count, and you will be penalized for lateness for every day between the due date and the date you upload your assignment to csman.** If you try to email your assignment to us, you will only annoy us, so don't do it! We don't care that you really did the assignment on time; if it isn't in csman by the due date, it's late and you will incur late penalties.

4. You should have read the general documents on the course website regarding course policies, especially the pages entitled "Administrative information" and "Collaboration policies". The last one is particularly important, because if you violate the collaboration policies, you may get sent to the Caltech Board of Control, which is never pleasant.

5. If the assignment due date is almost here and you still aren't on the course csman page, you should have emailed the instructor explaining the situation. As long as you have successfully filled out the signup sheet, adding you to the course csman page is very easy. If you haven't filled in the signup sheet, you should do that first and then email the instructor.

If you ignore any of these steps (like, say, if you just skipped reading this section), then you are liable for all late penalties that may occur as a result of your negligence. **We will not waive late penalties for negligence on your part.** However, if there is a problem that is clearly not your fault, we will take that into account.

What to hand in

All of your code should be saved to a file named `lab0.ml`. This file should be submitted to csman. For this assignment, all you will be doing is setting up a Linux virtual machine and installing the course software. There are a number of questions you will have to answer so we can verify that you actually did go through all the steps; answers to these questions should be written in OCaml comments in the `lab0.ml` file. Please indicate in a comment what problem any piece of code in your submission refers to. For instance:

```
(* QUESTION 1 *)
...
```

refers to the first question.

Part A: Setting up a Linux virtual machine

The CMS lab

The CMS lab computers (known as the "CMS cluster") are located in room 104 of the Annenberg building. This is the same room in which lab sections are held. In the past, many students have used the CMS cluster computers to write their assignments for CS 4 and other classes. However, these computers tend to break down a lot and don't always have the right software installed, so **we very strongly urge you not to use the CMS cluster computers for anything and to use your laptop instead when writing your code.** This will be much less frustrating for you. The rest of this section will explain how to do this. Note that we still very strongly recommend that you come to the lab sections (bringing your laptop along!), since the teaching assistants will be there to help you with any (course-related) programming problems you may encounter.

OCaml

In this course, we will be using the OCaml programming language. Fortunately, the OCaml language implementation works well on all major operating systems and is of very high quality. Note that we will be using the most recent version of OCaml in this course, which as of this writing is version 4.04.0. Earlier versions may also work, but there may be slight differences or missing features that we will use.

There is an "impedance mismatch" between the textbook we will be using (SICP), which uses the Scheme programming language, and the OCaml language we will be using for our assignments and lectures. This may confuse you a bit when you are reading the book, but we are doing it for very good reasons. OCaml is more readable, more modular, more type-safe, and more advanced than Scheme. It is also much more heavily used in real world applications, OCaml code executes much faster than Scheme code, and OCaml's type system means that your code will have far fewer bugs. Although OCaml is related to Scheme in a conceptual and historical sense (both languages are "mostly functional" and both are ultimately descendents of the Lisp programming language), the two languages are otherwise very different. We think that you will find learning OCaml to be fun and well worth the time you spend on it! If you read the book (which we recommend), you will probably find yourself picking up Scheme as well; there's nothing wrong with that. Scheme is a good language, but OCaml is a better one :-)

Note that one consequence of the Scheme/OCaml mismatch is that even though we will assign a lot of problems from the book, we will translate the problem description and the example code from Scheme to OCaml. We will also provide links to the book problems for completeness, but you shouldn't need to refer to them.

The course virtual machine

Instead of providing detailed instructions on how to install OCaml on every conceivable operating system and computer that you may have, we have chosen a simpler solution. This section will show you how to set up a virtual machine (VM) running Ubuntu Linux (technically, it's running the Lubuntu flavor of Ubuntu, which is a lightweight distribution). This VM has been pre-loaded with OCaml and all the development tools that you will need. The hardest part of this will be downloading the VM image, which is quite large (around 2.6 gigabytes). Once this is done, setting it up should be straightforward. We will include some questions for you to answer so we can check that your VM is set up properly and to verify that you are actually doing this. Your answers will be graded!

Of course, it's possible to run OCaml in any of the major operating systems (Linux, Windows, and Mac OS X); however, we are only going to support the version of Linux you will install here, so please don't ask the TAs to help you install OCaml on your Windows or Mac OS X computer. (You can do it yourself if you like; we aren't forbidding it, just asking that you not ask the TAs to do it for you or to help you if things go wrong in that case.)

Some features of OCaml (like the opam package manager) may not work well on Windows, which is another reason we have chosen to do things this way.

One thing must be made crystal-clear at this point. **These instructions assume that you will be installing a VM on your own computer. Do not, under any circumstances, attempt to install a VM on one of the CMS cluster computers!** This is not supported and will probably make the system administrators very angry, because it will waste a *lot* of disk space.

Also, if you find the VM setup process to be long and boring, take heart: you only need to do it once, and then you can use it as long as you want (even after this course is over).

Hardware requirements

In order for this to work, you should have a laptop (or desktop) computer running either Microsoft Windows, Mac OS X, or some version of the Linux operating system. Ideally your computer and its operating system are not too old (say, not more than 2 years old) and the computer has at least 4 gigabytes of RAM (random access memory). You should also have at least 20 gigabytes of free space on your computer's hard disk. Note that Chromebooks (at least, most Chromebooks) are not viable for installing virtual machines. If you don't know whether your laptop's specs are good enough, you need to find out – all programmers should know what operating system they are running and how much RAM and disk space they have on their computers. If your computer's specs aren't good enough to install the VM, you need to get a better laptop!

Downloading and installing VirtualBox

You will be setting up a virtual machine (VM), and in order to do this you will need software that manages and runs virtual machines. A virtual machine (at least in this context) is an operating system that runs in software (hence "virtual") inside another operating system. The outer ("host") operating system (which is the operating system of your computer, running directly on the computer's hardware) effectively simulates the hardware that the inner ("guest") operating system (running in software) sees. In fact, the guest operating system is not aware that it is being run as a virtual machine, though it will probably run a bit more slowly than it would if it were running on bare hardware.

The software we will use to run our VM is called **VirtualBox**. You should go to this page and download and install whichever version of VirtualBox runs on your operating system (just the "platform packages" download). For instance, if your laptop runs Microsoft Windows, you should download the version for Windows hosts. If your laptop runs Mac OS X, download the version for OS X hosts. If your laptop runs Linux, you should install VirtualBox using the package manager of your distribution; install the "virtualbox" packages but not the "virtualbox-guest" packages. Ask the TAs if you're not sure how to do this, but if you're already a Linux user, you probably already know how to do this.

Once you've downloaded the installer for your operating system, just click on it to install VirtualBox (unless your host operating system is Linux; see above). Both Windows and OS X will generally give you some grief about installing software from an untrusted source, and you may need to change some settings to allow this. Ask the TAs if you need help doing this. Other than that, the installation should go smoothly.

Downloading the course VM

Download the course VM from this link. This will place a file called "CS4_2017_VM.ova" in your `Downloads` directory. **This is a large file!** Therefore, we **very strongly recommend** that

1. you plug your laptop into an AC power supply while downloading, and
2. you use a wired Ethernet connection to do the download, or (if this isn't possible) use the fastest Wifi connection you have. Note that downloading late at night or early in the morning may give you better

results due to less traffic.

A word of advice: *please* don't try to download the VM over the "Caltech Guest" account! This is a *very* slow Wifi connection.

Note that if you don't download the entire file, you won't be able to proceed, so make sure the download continues until completion. If after several tries you still can't get it to work, ask your TA for help. If necessary, we can copy the VM file onto a flash drive if you give us the flash drive.

Once this is completed, pat yourself on the back. You have finished the hardest part of the assignment!

Installing the VM

1. First, start up VirtualBox. (You should be able to find it in the program menus, or you can use the terminal command line if you're using Linux (the program name is `virtualbox`.) A smallish window should pop up.

2. Go into the File menu and select "Import Appliance". (The file you downloaded is called an "appliance" for some reason.) You will get a dialog box titled "Appliance to import". To the right of the text entry field there is an icon which you should click; it will allow you to navigate through your file system to find the VM file you downloaded. Do so. (The file will probably be in a directory called "Downloads" under your home directory.) Once you've found and selected the VM file, click "Continue".

3. Another dialog box will come up called "Appliance settings". Don't worry about this; just click "Import" at the bottom.

4. Wait. This will probably take a few minutes.

If you've done this correctly, VirtualBox will return to its home window and you will see a new VM listed on the left side called "CS 4 2017 VM". You're ready to go!

Starting the VM

Select "CS 4 2017 VM" in the VirtualBox window and double-click on it (or hit the "Start" arrow at the top part of the window). The VM should start up.

After a while, you will see a login window, which may be smaller than the entire screen or not. The login name is preselected to be "Student". That's you! The login password is the same as the enrollment key for the course Moodle site ("algebraic"), which should be easy to remember. Type that in, hit return, and you will log in. (Feel free to change the password if you prefer something else.)

Now would be a good time to mention one annoying thing about VirtualBox. It has a tendency to "capture" the mouse, which means that when you click on the mouse inside a VM window, the mouse pointer won't exit it (or it will but you'll still see a duplicate pointer inside the window). There is a key you can press that will "release" the mouse and let you use applications outside of the VM. This is called the "Host key". On the Mac the Host key is the left Command key, and on Windows and Linux it's the right Control key (you can change the key binding if you really hate it). It's also listed in the lower-right corner of the window the VM is in (if the window isn't maximized). You should practice tapping the Host key in order to get out of the VM (ask a TA if you are having problems with this). Note that you may not need to do this on some systems; clicking outside the window may be sufficient. Clicking inside the VM window will put you in the VM again.

Once you have logged in, the VM should take up the entire window and you will see a gray background and a toolbar at the bottom. If it isn't maximized, hit Control-f (Command-f on a Mac) and it should maximize to take up the entire window. If that doesn't happen, something has gone wrong and you should see a TA.

Look at the toolbar. The important things to note are:

1. the menu button at the extreme left, which brings up a menu from which you can launch any program you like, shut down the system, *etc.*, and

2. some icons to the right of the menu button. These are program launchers. There is one for the file manager, one to launch a terminal, and one to launch a web browser. We are mainly going to be concerned with the terminal.

Click on the terminal icon, and a terminal should come up. Double-click on the terminal window's title bar to maximize it if you like.

QUESTION 1

1. At the terminal prompt, type: `uname -r` and hit the return key. (Another way to say this is to "Enter the line: `uname -r`".) What is the response? Write it down as an OCaml comment in your assignment submission.

2. Now enter the line: `which gcc` and write down the response (if any) in another comment.

3. Now enter the line: `which ocaml` and write down the response (if any) in another comment.

You can exit from the terminal by entering `exit` at the prompt or hitting Control-d.

At this point, you might think you're done, but there are still a few steps to go. First, you need to learn how to update your system. Most Linux distributions, including Ubuntu, are continually being updated, and it's important to make sure your system is up-to-date, because updates contain bug fixes and fix security holes. There are actually a few different ways to update your system. When you log in, if you need updates, a graphical program called `Software Updater` may already be running and will tell you that you need to update your system. (The window may be minimized; if so, click on the entry in the task bar at the bottom to bring it up.) Click on `Install now` to update the system. A window will pop up asking you for your password. You should enter this and then hit the `Authenticate` button, and then the system will start updating. If there are a lot of updates, this may take a while.

Once you're done, you need to restart the VM. A window will come up asking if you want to restart now or later; press `Restart now` and the system will restart. After the system restarts, enter your password to log back in.

The graphical installer doesn't always update everything, so you should also learn how to update the system from the terminal command line. This is actually just as easy and very powerful once you know what you're doing. To do this, first start up a terminal as described above. Then enter the following commands (the `$` is the terminal prompt; don't enter that):

```
$ sudo apt update
[enter your password]
$ sudo apt upgrade
```

You should only have to enter your password once. After `sudo apt update`, a bunch of stuff will be printed in the terminal, none of which you need to pay attention to. After `sudo apt upgrade`, the system will list all the packages that can be upgraded and offer you a yes/no choice as to whether to proceed or not. Hit the return key (which selects yes) to start updating. Also note that the terminal prompt is more complicated than just `$`; by default it also contains your username, the computer's name and the current directory. We will always use `$` in our examples, but never type the `$` into the terminal! It's possible to change the prompt to whatever you like; see any Linux tutorial for more on this.

Note that you shouldn't try to simultaneously update using the graphical updater and the terminal updater, as whichever one is invoked first will prevent the other one from running. Once you're comfortable with updating via the terminal, there's really no need to use the graphical updater anyway. Ideally you should update your computer about once a week to make sure everything is current.

Some simple customizations and stuff to know

Now you have a fully-functional and up-to-date Linux system installed as a virtual machine. You will see a toolbar on the bottom of the screen with a few panel "applets" on it to control things like volume, to show the time, to allow you to go to a different virtual desktop (called a "pager") to bring up the shutdown/restart window, *etc.*; these are on the right-hand side. On the left-hand side there is the menu button and launchers to start up various programs.

Click on the menu button and then select "Preferences". This will bring up a sub-menu where you can configure the system as you like. For instance, if you hate the plain gray background, select "Desktop Preferences" and pick a different color or some wallpaper image (there isn't much of a selection, unfortunately). You can configure other things too; bring up the different menu items and look around.

You will be working mainly in the terminal, so you need to be able to launch terminals quickly. In addition to clicking on the icon launcher in the toolbar, you can also just hit the key combination Control-Alt-t to bring up a terminal.

Since you'll be working in the terminal so much, it is worth spending a couple of minutes configuring the terminal to be as comfortable as possible. You may want to change the terminal's background color, the text color, or the font used for the text to suit your preferences. To do this, launch a terminal, click on the `Edit` menu and then select the `Preferences` option. This will bring up a window in which you can select a number of preferences. To change the font, click on the `Terminal Font` pulldown menu. You can choose between a number of fonts, each at a number of different sizes. Pick whatever you find most comfortable, then click `OK`. If you've changed the font, you will see the change. If you want to change the background or text colors, click on the `Background` or `Foreground` items in the window and select whichever colors you like. Note that `Foreground` is the text color.

Also, note that the terminal program can create multiple terminals or multiple tabs in a single terminal. Tabs are very useful when doing multiple distinct things. Create a new tab by typing Control+Shift+t. Create a new terminal window by typing Control+Shift+n. (These actions can also be done from the terminal's menu.) Type Control+d in a terminal window to close it, or just type `exit`.

There are other terminal configurations you can set, but this is enough for now.

Some people like to remap their keyboard in various ways. The most common such mapping is changing the "caps lock" key (to the immediate left of the "a" key) to mean "control", and changing "control" to mean "caps lock". This is easy in some Ubuntu flavors, but not as easy in Lubuntu. If you want to do this, first see if you can do it in your host operating system (*e.g.* Windows or Mac) and then see if it Just Works on your virtual machine. This will often be the case, but if not, you can look at the instructions on this web page. The article says to create a `.Xmodmap` file, but doesn't specify where it should go. You should put it in your home directory (here, `/home/student`). Also, some of the language in the web page is a bit offensive; sorry about that! (I didn't write it.) Other keyboard customizations can be done in a similar way. If you want to remap the "command" key (Apple; Windows calls it the "super" key), this link may help.

Configuring OCaml

You don't have to configure OCaml! Everything should be set up for you. Start up a terminal and type this:

```
$ opam list
```

You should get output like the following:

```
# Installed packages for system:
base-bigarray            base  Bigarray library distributed with the OCaml compiler
base-threads             base  Threads library distributed with the OCaml compiler
```

```
base-unix               base  Unix library distributed with the OCaml compiler
... [many more packages]
```

`opam` is the OCaml Package Manager. We have tried to preload all the OCaml packages you will need for this course, but if you need to install new ones, it will be very easy to do (we'll tell you how if/when the time comes).

Text editors

You are free to use whatever text editor you like to edit your code as long as it outputs *plain text* and not *e.g.* Rich Text Documents, PDFs or Word documents. We've installed several text editors into the VM, including `emacs`, `vim` and `atom`. Of the three, `atom` is the easiest to pick up, so if you don't already know `emacs` or `vim` that is the one you should choose. All three editors feature syntax coloring for OCaml source code, which is very helpful.

To start up *e.g.* `atom`, type this in a terminal:

```
$ atom&
```

Then the editor window will pop up. The `&` is so you can continue to type commands into the terminal while `atom` is running.

Using OCaml

There are only a few things you need to know in order to use OCaml effectively.

1. On your new Linux system, you can start OCaml by opening a terminal and typing

   ```
   $ ocaml
   ```

   at the terminal prompt. This will bring up the OCaml interactive interpreter, which is a good environment for experimenting with the language and for testing code you've written.

2. One tool that makes it easier to use OCaml interactively is the `rlwrap` program, which is a line editor. (You installed this earlier.) It makes it possible to easily recall and edit previously-input lines of text by using the up and down arrow keys. We have already configured OCaml to use `rlwrap`. To see what it can do, start up the OCaml interpreter:

   ```
   $ ocaml
   ```

   and type the following commands (one per line, hitting the return key at the end of each line):

   ```
   # Printf.printf "hello\n" ;;
   # Printf.printf "goodbye\n" ;;
   ```

   Note that the OCaml interpreter prompt is the hash sign (`#`); don't type that! These lines should, when evaluated, print the words `"hello"` and `"goodbye"` respectively. Once you've done this, you should be able to recall either line by using the up arrow key. For instance, you can hit the up arrow key once to get the line:

   ```
   Printf.printf "goodbye\n" ;;
   ```

   and edit it so that it says:

   ```
   Printf.printf "hasta la vista\n" ;;
   ```

   When you hit return, it should print out `"hasta la vista"` on a separate line. `rlwrap` makes line editing much easier. You can also use Control-a to get to the beginning of a line you are editing and Control-e to

get to the end. Control-l (lower-case L) clears the terminal and starts you at the top.

By the way, in addition to printing out the messages you wanted, you'll also see the line:

```
- : unit = ()
```

printed out after hitting return. This is OCaml telling you the type of the result. In this case it's the `unit` type, which is what `printf` returns. We will talk about this in class.

Exit the interpreter by typing Control-d (this is just like Python).

3. One confusing aspect of OCaml is that when entering code in the interactive interpreter, nothing gets evaluated until you enter a double semicolon (`;;`) followed by a carriage return. What this does is tell the OCaml interpreter "Hey, I'm done entering code, so take the code I've entered and evaluate it.". The advantage of this is that you can enter code that spans more than one line without having to enter line continuation characters at the end of lines (like you have to do in *e.g.* Python). The confusing part is that it's almost never necessary to enter the double semicolons when writing OCaml code in a file. This is a very common mistake of beginning OCaml programmers! It usually doesn't cause any harm, but it looks bad, so don't use the double semicolons when writing OCaml code in files. **We will take style marks off if you violate this guideline!**

4. The usual way of writing code using OCaml is to write it in a file and then load the code into the interactive interpreter to test it. The interactive interpreter has several commands that are used for this, all of which begin with the hash sign (`#`). The simplest one is the `#use` command, which loads up some OCaml source code from a file, compiles it, and runs it. For instance, open up your text editor and type the following into a file called `"test.ml"`:

   ```
   let f x = 3 * x * x + 4 * x + 5
   ```

   (Note, by the way, that we didn't use the double semicolons, since this is OCaml code in a file!). Save the file, then start a terminal and navigate to the directory containing that file. Then start up the OCaml interpreter:

   ```
   $ ocaml
   ```

   Inside the interpreter, type the following command:

   ```
   # #use "test.ml";;
   ```

   The first `#` is just the OCaml prompt and shouldn't be typed. If you've done this right, OCaml should respond with:

   ```
   val f : int -> int = <fun>
   ```

   What this means is that OCaml has compiled the code in the file `test.ml` and is printing out the type signature of all functions it has found there (in this case, there is only one, `f`). Test that it works by typing the following:

   ```
   # f 10 ;;
   ```

   OCaml should reply with:

   ```
   - : int = 345
   ```

   By the way, the "`-`" to the left of the colon just indicates that the value that is printed is not a named value (unlike `f`, which we saw above).

5. Most of the time, we will compile OCaml code outside of the interactive interpreter by using the OCaml compilers `ocamlc` and `ocamlopt`; we will discuss this later.

QUESTION 2

Use your text editor to open up the `test.ml` file you created above and add this line to it (below the line defining the `f` function):

```
let g x y = x *. x +. y *. y
```

Note that you shouldn't have semicolons (single or double) anywhere in the `test.ml` file. Then save the file, start up the OCaml interpreter, and type:

```
# #use "test.ml";;
```

at the OCaml prompt like you did before (not typing the `#` prompt, of course). What does OCaml print in response? Again, write your answer in an OCaml comment.

Shutting down

To get out of the VM, select "Logout" from the VM menu and click on "Shutdown". This will exit the VM.

OK! Now you are all set up to run OCaml on your VM. If things didn't work out as described above, see a TA.

Part B

There is no part B. You are done!
