COSC–111 Introduction to Computer Science I
Lab 01: A First Program
Due: Friday, February 8, 8.59am

1 Getting started

The following steps will get you started with using the lab: you will be introduced to the programs and tools that you will need to complete the lab and project assignments. Please ask questions (raise your hand and someone will come to you).

1. **Login to your workstation:** Our lab is full of Windows desktop computers. These are run by our Information Technology department, and you must begin by logging into them using your college username and password. These workstations won't do much themselves: they will primarily be used to connect to different systems on which the real work will happen.

2. **Login to a server:** The computer systems that we will use for our projects are `romulus.amherst.edu` or `remus.amherst.edu`, (henceforth, `remus/romulus`), which are UNIX (Linux) systems.[1] To use these systems, you must login to them from your computer, using software known as an *X11 Windows Server*. Installing and starting this software is modestly different from one type of computer to another:

   - **Windows:** Follow these instructions to install and run *VcXsrv* for the first time.

     (a) **Install:** (**If you are working on one of the lab's workstations, skip this step.**) Go to the web site for *VcXsrv* (`https://sourceforge.net/projects/vcxsrv/`. Click the large and obvious button labeled, *Download*. Your browser will download a file named something like:
     `vcxsrv-64.1.20.1.4.installer.exe`
     When this file is completely downloaded, run it. The installer will ask permission to install the software; click *Yes*, and accept any defaults about where to put the program, etc.

     (b) **Download configuration(s):** Go to Prof. Kaplan's *VcXsrv* web page (`https://sfkaplan.people.amherst.edu/VcXsrv`), where there are *XLaunch configuration files*—one for connecting to each server. Download one or both by **right-clicking** on it and then selecting *Save link as…*. We recommend saving these to your `Desktop` folder for easy access.

[1] These servers are identical in all but name: no matter which one you are using, you will see the same files and use the same programs. There are two of them simply to spread the load of so many students connecting at one time.

(c) **Launch:** On your desktop (or wherever you saved them), double-click on one of the XLaunch configuration files to launch it.

(d) **Connect:** The first time you connect to each server (and only the first time), you will see a window asking you if you want to accept the server's credentials/keys. Type `y` and press enter.

A window will then prompt you for your **username**. Enter it. Note that (somewhat strangely) asterisks will appear, hiding what you're typing.

One more window will prompt you for your **password**. Enter it. Again (and this time, more sensibly), asterisks will appear in place of what you type.

After all of these steps, you will be presented with an *xterm terminal window*, in which you will be presented with a *shell prompt*. Continue onto the next step.

- **MacOS:** Follow these instructions to install and run *XQuartz* for the first time:

(a) **Install:** Go to `https://www.xquartz.org/`. There, download the installer shown under the header *Quick Download*, named something like:

```
XQuartz-2.7.11.dmg
```

Open this file, which will start the installer. Follow the instructions and accept the default options to install *XQuartz*.

(b) **Launch:** When the installation is complete, search for `XQuartz` (or go to Applications, and then to Utilities) and run it.

From the menu bar, click on the *Applications* menu, and then select *Terminal*. A *terminal window* will appear, and within it there will be a *shell prompt*, at which you can type commands.

(c) **Connect:** Enter the following command to connect to one of the servers. Replace `username` with **your username** before the `@` symbol; you may connect to `remus` as shown, or replace that with `romulus` to connect with that server:

```
$ ssh -Y username@remus.amherst.edu
```

You will be asked a very technical question about a host key. Type `yes` and press enter.

When prompted for your **password**, enter it. Nothing will appear as you type, hiding your password from anyone looking at your screen.

In your terminal window, you will see a new shell prompt that shows that you are connected to the server that you chose. Move onto the next step.

3. **The *shell*:** In your *terminal window*, you will see a prompt presented by a *shell*—a program to which you type commands to the system. The shell interprets your commands and launches the program your request. You will use it to start a *text editor* to see and alter your source code, to run a *compiler* that checks and transforms your source code into a program that can run, and to *execute (run)* the programs that you create.

   As a first command, just to try things out, try the `whoami` command. That is, at your prompt, type that command and press enter; the program's response (which should be your username) should appear. It would look something like this:

   ```
   [mriondato@remus ~/Desktop]$ whoami
   mriondato
   ```

   Notice that everything up to and including the dollar-sign (`$`) is the *shell prompt*—what the shell prints to the terminal to show you that it's ready for a command. You type only what follows. Henceforth, in these lab/project documents, I will use just the dollar-sign to indicate the prompt, even though your shell likely prints some stuff (e.g., your username, the server to which you're connected) prior to that dollar-sign.

   As a second command, see who else is connected to the server with the `who` command, which would look something like this:

   ```
   $ who
   root      pts/0  2018-10-04 10:54 (sngtools03.amherst.edu)
   sfkaplan  pts/1  2019-01-25 09:42 (temp6.cs-res.amherst.edu)
   lamcgeoch pts/5  2018-11-26 19:00 (:279.0)
   mriondato pts/22 2018-11-08 21:03 (:225.0)
   ```

4. **Make a directory:** When you first login and open a terminal window, you will be working in your *home* directory.[2] Your shell prompt (likely) shows you that you are in your home directory with the part that reads, `~`. The *tilde* (`~`) is used by the shell as a shorthand for your *home directory*

[2] *Directories* and *folders* are the same thing. Your home directory here is just like your `Documents` folder on your own computer.

To create a directory for your work for this lab, use the `mkdir` (**m**a**k**e **dir**ectory) command, like this:

```
$ mkdir lab-1
```

Notice that this command produces no output: *no news is good news* with many commands.

Next, use the `cd` (**c**hange **d**irectory) command to make that new folder the shell's current one, like this:

```
$ cd lab-1
```

You should notice that your shell prompt changes, showing the current directory as something like: `~/lab-1`

5. **Get some starting source code:** Use the following command to obtain a sample Java source code file:

   ```
   $ wget -q -O Messages.java http://bit.ly/111s19L01
   ```

   To ensure that you have copied the file into your `lab1` subdirectory, use the `ls` (***l**ist **s**irectory*) command. The `-l` part (it is "dash ell", not "minus one") of the command indicates that you want to list the directory using the *long format*:

   ```
   $ ls -l
   ```

   You should see an output that looks something like this:

   ```
   total 4
   -rw-r----- 1 mriondato mriondato 623 Jan 18 10:55 Messages.java
   ```

   There is a lot of information displayed, and we are not going to discuss it, but let's just say that the shell is a very powerful tool to get information about the system, and in general to perform all kinds of operations.

You have now set yourself up to work with some Java source code. Move on to the next section.

2 Your assignment

Perform the following steps in order to see, use, and change your code:

1. **Examine the source code:** Run *Emacs*, a programming text editor, to examine the `Messages.java` file. In the following command, be sure to include the trailing ampersand (&), causing the text editor to run in the *background*—that is, to run while allowing you to enter more commands:

   ```
   $ emacs Messages.java &
   ```

   A new window will open, showing you a handful of lines of Java code (along with *code comments*—lines that begin with two forward slashes (`//`)—that provide some human documentation and commentary. This code shows a program named `Messages` that will start on the line beginning `public static void main (String[] args)`, ask the user a question, read in the user's answer, print a message, and then end. For now, examine the code, but leave it as is.

2. **Compile the code:** The source code must be translated it into a form that the computer can execute. Leaving your *Emacs* window open, click over to your terminal window again. In it, use the following command to *compile*—that is, translate into machine code—your source code:

   ```
   $ javac Messages.java
   ```

   In this case, **no news is good news**. That is, if the computer simply presents the shell prompt to you after you issue this command, then **the compilation succeeded**. The compiler—the `javac` program—will print messages into your terminal window only if it was unable to translate your program. Given that you have not changed anything from the original code, it should compile without error. The compiler generated a file `Messages.class` containing the machine code for your program (check using `ls` that a new file is now in your directory).

3. **Execute your program:** Once you have successfully compiled your program, it is time to run it and see what happens. Go to your shell window and issue this command:

   ```
   $ java Messages
   ```

   The first question that you saw in the code should be printed into your terminal window. Type an answer in the terminal window, then press enter. You should now see the second question in the code printed in your terminal window. Continue answering the questions until you see the final set of messages print to the screen. At this point you should again see the prompt in your terminal window.

4. **Change something:** Alter your program slightly, and then test that your alteration did what you expected. Specifically:

   (a) **Edit:** Go back to your *Emacs* window. **Add your own print statement or two**, using the existing print statements as templates. Be sure to **save** your changed source code. You can do this either by clicking the File menu and then Save, or by using the *Emacs* shortcut Ctrl+x Ctrl+s (that is, hold down the Ctrl key and hit x, release both buttons, then hold down the Ctrl key again and hit s).

   (b) **Compile:** Issue the `javac` command to your shell, just like you did above. If there are error messages, you will need to return to *Emacs*, fix whatever is incorrect, save the changed code, and try to compile with `javac` again. Repeat until your code compiles with no error messages.

   (c) **Execute:** Issue the `java` command to your shell, just like you did above. Your program should run, this time printing the additional message(s) that you added into the code.

5. **Change something else:** Make some more changes to the program, using the code we saw in class this week as examples. Here are some ideas that you might want to try:

   (a) Ask the user more questions. Use the existing code as an example for how to read the user's response from the keyboard.

   (b) Declare some variables of type `int` or `double`, do some arithmetic, and print the results.

   (c) Read numeric input from the keyboard. You can do this using the `keyboard.nextInt()` and `keyboard.nextDouble()` commands.

   After each change that you make, be sure to **save** your work, **compile** your program using the `javac` command in the terminal window, and **run** your program using the `java` command in the terminal window. It is in your best interest to compile and run your code often as you make changes, as this will allow you to catch errors early.

Ta-da! You've just done some programming. You have also begun to use the tools on which your work will depend in this course: remote desktop, remus/romulus, the shell, the *Emacs* text editor, the `javac` compiler, and the `java` executor.

3 How to submit your work

Submit your modified `Messages.java` file. You may use either of the following two methods to use the CS submission system:

- **Web-based:** Visit the submission website (https://www.cs.amherst.edu/submit). If you did your work on your own computer, open this link in your computer's browser; if you did your work on `remus/romulus`, then open this link in a browser opened within *remote desktop*.
- **Command-line based:** Use the `cssubmit` command at your shell prompt. This command works only while connected to `remus/romulus`. Type:

  ```
  $ cssubmit Messages.java
  ```

  and then follow the instructions to complete your submission.