CSci 1113
Lab 00

First, find a partner (maybe)
Remember one goal of the labs (unlike the individual homework assignments) is to give you a chance to work closely with others. Programming in industry is often done in teams; therefore, being able to program collaboratively is an important skill. We encourage you to do the lab in pairs, but this is not mandatory. You will not need to work with the same student all semester; in fact we encourage you to work a different student each week, at least during the first few labs. If you have trouble finding a partner, the TAs can help.

Register your account

Prior to lab you should have registered your CSE Labs account. If you have not yet done so, use a web browser to visit http://www.cselabs.umn.edu/ and then click the "Account Information" link. Once you register your account it will take a while to be activated, so you will need to work with a partner with an already activated account.

Logging in

To log in from a machine in one of the CSE computer labs (such as KHKH 1-260) pick an available Linux machine and type your *username* and *password* in the login box.

Start geany

To start geany, you must first open a terminal. This can be done either by clicking this icon:

... or by pressing Ctrl+Alt+T. Once the terminal is open, simply type “geany” (without quotes) and then hit enter. Once geany is open, create a code file by selecting FIle -> New, or click this icon:

Finally, save this file with any name but you must use the extension “.cpp” at the end (without quotes). To do this select: File -> Save as, then navigate to where you want to save it (possibly creating a folder called lab0) then give it a name (such as: first.cpp).

Compiling and running

Copy paste helloWorld.cpp from the class website into the blank file you just saved. Then click this icon to compile:

... Then this button to run (it is not greyed out!):

Modifying the program

Change the program to say “I am done with lab” instead of “Hello, world!”. Run the program and show it to the TAs.

Logging out (DO NOT SHUTDOWN)

Below is a Linux tutorial if you wish to further familiarize yourself with the lab machines (or you are very bored). If you want to do someting else with your time, click the icon in the top right and select log-off. Do not restart or turn off these computers ever!

Introduction to UNIX and C++

Welcome to the first lab in CSci 1113. In this lab you will explore some computational resources needed during this course, and write a few short programs. The programming environment used in the labs runs under a variant of the UNIX operating system called Linux. This lab exercise will familiarize you with the process for creating C++ programs in the Linux environment.

Even though this lab description is long, much of it is explanation, and so you should be able to complete the essential parts of the lab in three hours.

Comment on this lab

Because this is your first lab, a UNIX tutorial is provided with commands you will need to know, as well as how to create, compile, and run a simple C++ program. Future labs will be similar to , which involves a number of short C++ programs.

Brief UNIX Introduction (adapted from C. Swanson, "Unix Tutorial")

The UNIX operating system dates to the early 1970's and remains one of the most efficient, robust and reliable pieces of software in use today. The "operating system" refers to the collection of programs that reside on your computer and provide the interface to the actual hardware of the machine. This involves maintaining and organizing data on external devices, loading and running application programs, etc. You are probably familiar with graphical interfaces to operating systems such as Microsoft Windows, Apple Mac OS and others (actually, Mac OS uses the UNIX operating system for its core functions). Our lab computers employ Linux, which is a version of UNIX, so you will need to learn a few UNIX essentials. As usual, you are encouraged to explore and learn more on your own! There is a wealth of UNIX information and tutorials available online. Moreover, there are many excellent textbooks such as *Just Enough UNIX, Fourth Ed.*, by Paul K. Andersen, McGraw-Hill, 2003.

The UNIX Terminal

Unlike many other computer systems you might have used in the past, you will be interacting with UNIX mainly using a *command line interface*: you will usually type commands into a terminal window along with required and optional *arguments*. Arguments are additional information that tell the computer what to do with the command. For example, if you want to rename a file, you need to provide both the UNIX rename command (`mv`), and the name of the file to be renamed.

To run programs and generally interact with the UNIX operating system in command line mode, you must first start a terminal window application by clicking on the terminal icon (generally located on the upper left side of your display -- it looks like a little computer monitor). Start the terminal now (seek help from your TAs if you need it or if the monitor icon is not visible).

The Command Line Interface
The operating system will prompt you for a command, using a short text string like:

```
username@machine%
```

where *username* is your UNIX user name and *machine* is the machine you are currently logged in to. The '%' symbol is there to let you know that the system is ready for another command. Commands can be edited while they are still being typed using the left and right arrow keys or the backspace key. *Important note*: UNIX commands must be typed *exactly*, and are *case-sensitive*. Case-sensitive means that upper- and lower-case letters are treated differently, for example, the rename command is `mv`, not `MV` or `Mv`.

Most UNIX commands are shortened names that describe what they do. For example, `cp` stands for *copy* and `mv` stands for *move*. If you ever forget what a particular command does or how it is used, you can use the `man` command to learn more. `man` stands for *manual* page and will tell you everything you need to know (and more) about a particular UNIX command. Simply type: `man` *command-name* [ENTER] to display the manual page, and press [SPACE] to scroll through the pages of information. You can even use `man` to learn about `man` itself! Try it:

`man man` [ENTER]

Files and Directories
You are probably familiar with the concepts of files and folders. A file is a collection of data that has a specific name associated with it. A folder is a special file that contains zero or more files or other folders. In UNIX, folders are called *directories*, but they are essentially the same thing. Files and directories allow users to store and organize their data.

File and directory names are case-sensitive, and special care should be taken when naming them. Avoid using spaces or special characters except for the underscore, dash, and period (and be very careful with underscores and dashes). Certain special characters are forbidden in filenames, but others are allowed and can cause problems. Also, avoid giving files the same name as UNIX commands.

Navigating the File System
All of your files and directories are contained within your *home* directory. When you first start a new terminal, your *home* directory will always be your (active) *working* directory. The working directory is simply the directory you are currently viewing, or working in. You can use the *print working directory* command, `pwd`, at any time to show you what your current working directory is:

`pwd` [ENTER]

You should see something like:

`/home/`*username*

This is the complete *path* from the first (root) directory to your current working directory. Each directory name in the path is separated by forward slash characters.

To list the names of all of the files and directories in your working directory, type the *list* command, `ls`:

`ls` [ENTER]

All file names will be listed; directories are displayed with an ending forward slash. Directories can be created and destroyed using `mkdir` (*make directory*) and `rmdir` (*remove directory*). Try making a new (sub)directory:

```
mkdir hello [ENTER]
```

You can change to a different working directory with the *change directory* command, `cd`. To change to your newly created `hello` directory:

```
cd hello [ENTER]
```

Now we will introduce some special names for directories. The double-dot (`..`) stands for the *parent* directory, which is the directory that contains the current working directory. We can use this to "move up" one level. Try

```
cd .. [ENTER]
pwd   [ENTER]
```

Also, the tilde (~) is shorthand for your *home* directory. We could have also returned to the home directory (from anywhere) by typing:

```
cd ~ [ENTER]
```

Now that we have returned to the home directory, we can remove our hello directory. Type

```
rmdir hello [ENTER]
ls [ENTER]
```

Note that `rmdir` will only remove empty directories (that is, directories that contain no files or other directories).

Creating C++ source code files

C++ is a *compiled* computer language. Generally, compiled languages execute much more quickly than *interpreted* languages, but this speed comes at a cost in development complexity. C++ source programs are not directly *executable* by your computer; you must first perform an intermediate *compilation* step that converts your C++ source program (text) into a form – *machine code* or *executable code* – that is suitable for loading and execution by the operating system.

To write a C++ program, you normally begin by using a *text-editor* to create a file with a `.cpp` extension, e.g., `test.cpp`. A text editor is a relatively simple program (application) that works like a stripped-down word processor. It allows you to enter/modify text and save the text to a file. There are a number of text editors available. We'll use the `geany` text editor for this lab assignment.

The file you create contains C++ statements and is known as a *source code* file. We (humans) are able to understand source code because it is vaguely like English, but the computer can't. Therefore we must first *compile* the source code using a *compiler*. There are many C++ compilers; the one used for this class is named `g++`. `g++` processes a source code file and creates a file the machine can run. If the compiler does not understand what you wrote in your source code it will complain by generating one or more *syntax* errors. You need to look at the error messages, identify the errors, and then use geany again

to fix your source code. Sometimes your program will compile correctly but will fail to execute due to a *run-time* error. An attempt to divide by zero is a common example. Finally, your program may compile and execute but produce incorrect results. This is due to one or more *logic* errors, and you will need to go back to the editor and fix your source code. In each case, you need to recompile the program and test it again until everything is okay.

First Program

Let's write our first program! It will be a short program that simply displays to the terminal

First, create a special directory for this lab. Enter the following:

```
mkdir mycode  [ENTER]
cd mycode     [ENTER]
```

The first command creates a subdirectory named `mycode` (Note: keeping separate labs and assignments in different subdirectories is a good idea to help keep your account organized). The second command moves you to that subdirectory.

Creating the source code file

Start the GEANY editor to create a file called `assign0.cpp`:

```
geany assign0.cpp [ENTER]
```

Once GEANY is running, you can enter source code. Figure out how to copy and paste the following program from this document to the geany window. Ask a TA if you have trouble. Note that computer languages are very precise and the compiler has no tolerance for errors! Every symbol in this program has significance and must be copied *exactly* as it appears (except for your name and the date in the *comment* statements). Note the use of semicolons in some of the lines.

```
// Assignment 0
// Type your names here
// Type today's date here
//
// This program displays to the screen
#include <iostream>
using namespace std;

int main()
{
```

```
    cout << "Hello!" << endl;
    return 0;
}
```

Next edit the file by replacing the second and third lines by your and your partner's names, and by today's date, respectively. Once again, ask a TA if you are unable to figure out how to do this. Note that simply typing or pasting text in the editor program does not actually *create* the file; you must explicitly *save* the text to the file. To save your changes, select "File" and "Save" in the pull-down menu. We'll use this program later, but for now it will provide an opportunity to learn some additional UNIX commands. Once you have finished typing in the program, save it and then close (quit) geany.

Some More UNIX

Before you compile and run the program, here is some more UNIX. Your `assign0.cpp` source code is now saved as a file within the `mycode` directory. Suppose we really had intended to put the file in a directory called `lab0` under our home directory. Making this change will give us an opportunity to practice some UNIX commands. If you aren't already there, change to your home directory:

`cd ~` [ENTER]

Now make a new directory named `lab0:`

`mkdir lab0` [ENTER]

and change your working directory to the `mycode` directory:

`cd mycode` [ENTER]

We'll use the *copy* command, `cp`, to copy files from one directory to another. The copy command takes the form

`cp` *source-file destination-file*

where *source-file* is the name of the file to be copied, and *destination-file* is where we want to put it. Use the list command to list all the files in the `mycode` directory:

`ls` [ENTER]

Now, copy the file using the `cp` command

`cp assign0.cpp ../lab0/` [ENTER]

The `../lab0/` tells the cp command to find the directory called lab0 in the *parent* directory, and copy the files there. Now use the *list* command to list the names of the files in the lab0 directory, to make sure they were copied successfully:

`ls ../lab0/` [ENTER]

Once you have verified that you copied the files correctly, you can delete the files in the `mycode` directory using the *remove file* command. Type:

`rm assign0.cpp` [ENTER]

Be careful! Unlike other computer systems, there is no "undelete" or "Recycle Bin" in UNIX. Once you delete a file, it's gone for good in most cases. Now return to your home directory:

```
cd ~ [ENTER]
```

Copying `mycode` files into the `lab0` directory was good practice, but we could have saved ourselves a lot of work by simply renaming the `mycode` directory! Try it yourself. Use the *move* command to rename a directory:

```
mv mycode  hello [ENTER]
ls [ENTER]
```

Change it back using:

```
mv hello  mycode [ENTER]
ls [ENTER]
```

Note that the `mv` command can be used to rename either files or directories.

Recall that if we want to delete a *directory* instead of a file we must use the *remove-directory* command: `rmdir`. Since our source code file is now in the `lab0` directory, we no longer need the `mycode` directory so let's delete it:

```
rmdir mycode  [ENTER]
ls  [ENTER]
```

The `mycode` directory should be gone from your home directory (and it's REALLY gone!)

Lastly, we will discuss two more convenient functions provided by the command line. These are *command history* and *tab completion*.

Tip: Command History (VERY USEFUL, USE THE ARROW KEYS)
You may have noticed already that pushing the up arrow in a terminal window brings up previous commands. You can use this to your advantage to recall any command you have typed in that terminal window. This is especially useful if you make a mistake in a long command. You can simply press the up arrow until you arrive at that command, and correct the error using the left and right arrow keys. Try it:

```
makdir hello [ENTER]
```

This should give an error. Press the up arrow once to recall the command. Then press the left arrow until the cursor is on the 'k'. Then press backspace to remove the erroneous 'a'. Finally, press [ENTER] to retry the command. The cursor does not need to be at the end of the command; the terminal will take the whole line as the command regardless of cursor position.

Tip: Tab Completion (ALSO VERY USEFUL)
Often, you will have long filenames that are difficult to type. Tab completion will save you typing. You can simply type the first few letters of a file or directory name, press the tab key, and UNIX will try to complete the name for you. Try this:

```
cd ~ [ENTER]
cd h [TAB]
```

The terminal should "fill in" the rest of hello. But what happens when multiple files start with the same characters? Try this:

```
mkdir hydrogen [ENTER]
cd h [TAB]
```

The terminal should display the names of all files and directories that start with h. In this case, hello and hydrogen both qualify. Type

```
y [TAB]
```

and the terminal should fill in the rest of hydrogen.

You've now learned to use several basic (and important) UNIX commands for manipulating files. If you are still unsure of what you are doing, discuss it with one of your lab TAs before continuing. You may also find it useful to explore the wealth of information and tutorials available on the Internet. Simply search for "unix tutorial" using Google or any other online search engine.

Check
To check you knowledge of UNIX, both you and your partner should individually make a list of six to ten UNIX commands. Try to do this without looking up the commands in this lab write-up. When both of you are done, compare lists. Also make sure both of you know what the commands on both lists do.

Compiling C++ programs

As described earlier, the process of creating a C++ program involves creating a *source-code* file using an editor of some sort, and then converting the source-code file into an executable program using the C++ compiler. The compiler will create an executable file named `a.out` that the computer can run.

To compile the `assign0.cpp` program you created earlier, make sure you are in the `lab0` directory and type the following on the UNIX command line to run the g++ compiler on `assign0.cpp`.

```
g++ assign0.cpp [ENTER]
```

If you entered the `assign0.cpp` program correctly, then g++ should compile it without any error messages and produce an executable program file named `a.out`. Type:

```
ls [ENTER]
```

and make sure both `assign0.cpp` and `a.out` are in the directory listing.

If the compiler produced one or more error messages, then you probably made a typing or copying error. Check the provided program again and make sure your program is *identical* to it. If you find a place where they differ, run geany again (`geany assign0.cpp`), correct your program, resave it, and recompile it (`g++ assign0.cpp`).

Executing your program

Once the program compiles, run it by typing the executable file name preceded by `./` :

```
./a.out
```

This should display Hello! to the screen. What happens if you remove the "<< endl" part?

Now show the following to one of your teaching assistants:

A. A listing of your program using the `cat` command (type `cat assign0.cpp`)
B. The compilation of your program using the `g++` command
C. A test run of your program.

Logging Out (DO NOT TURN OFF THE COMPUTER)

When you are done using the computer type:

`logout` [ENTER]

or,

`exit` [ENTER]

to log out of your account. All of the windows you were using should disappear and the login screen should reappear. *Remember to logout at the end of each lab.*