Lab Assignment 1:

Zeus, Hello World, and Blackboard Submissions

Due: Friday, September 7 at 11:59 PM

Description:

The purpose of this assignment is to introduce you to creating and compiling C programs in a Unix environment. You will also learn a few Unix commands and learn the procedures to submit your assignments using Blackboard. There is a lot of material to cover in this lab (most later labs won't have nearly as much text). Fortunately, the instructions are mostly step by step, and should not take much effort. However, you **must not** blindly follow the procedures without understanding what you are doing. The procedures you learn in this lab will be used **repeatedly** throughout the entire semester and you will best serve yourself by learning them **thoroughly** this week.

You will first login to zeus, the computer on which all your labs and projects must run. You will create a directory in which to work on this lab, then create, compile and test a C program using both the *make* command and the gcc compiler on Zeus. You will then follow instructions to create a typescript of a session where you demonstrate that your program compiles and runs without errors. Next, you will create an archive file (tarfile) of your working directory. Finally, you will copy this file to your local computer which you will then submit to Blackboard as Lab 1. Nearly all future programming assignments (Labs and Projects) will use these procedures, so be sure to understand **completely** what you are supposed to do. **Note that the failure to follow these procedures and file naming conventions on this and future assignments will result in a loss of points and/or no grade for the assignment.**

Things you should know how to do after completing this lab:

- Log into zeus.ite.gmu.edu
- Copy files from zeus to your local computer
- Create directories on a UNIX system and make them your current working directory
- Create and compile a simple C program on a UNIX system
- Basic UNIX commands (ls, cd, mkdir, rm, pwd)
- Create a tarfile on a UNIX system

General Comments:

There are many different compilers and operating systems available for you to create programs using the C programming language. However, in order to make grading in this course consistent, we will be requiring that your programs work correctly on zeus.ite.gmu.edu. It is **strongly** recommended that you become familiar with the *vi* or *vim* editors on zeus so that you are not

spending an inordinate amount of time transferring files back and forth between your local computer and zeus. Supplemental information on using vi and vim can be found in the CS262 Course Content folder on Blackboard. Although you may edit and develop programs on whatever system you wish, students who learn to use zeus exclusively for their labs and programs tend to gain more from the class and earn higher grades in the course. To this end, general instructions for assignments, such as how to compile and run programs, will only be given for zeus. Do not expect assistance from the instructors for code development on any other platform. Also be aware that programs that run correctly on a different system may not run the same way (or at all) on zeus. If you decide to use a system other than zeus for program development, be sure that you allow enough time to ensure your programs will run properly on zeus. **You will not be given extensions or credit for programs that work on another system, but do not work on zeus.**

Logging Into Zeus and Basic Unix Commands:

The first step of this assignment is to login to zeus. If you are on campus, you can login directly using the ssh secure shell program found on Windows machines, or if you are on a Unix machine or a Mac, you can bring up a terminal window and type:

```
ssh <username>@zeus.ite.gmu.edu
```

Replace the <username> in the command above with your username (Your username is your email address, up until the @gmu.edu. For example, if your email address is astudent@gmu.edu, then your username is astudent). After typing the command, follow the prompts to enter your password. Your password will be the same as your general GMU password (i.e. the password you use for your email).

If you are off campus, then to login to zeus you must either 1) use the VPN provided by GMU Tech Services or 2) login to mason.gmu.edu and from there, login to zeus.ite.gmu.edu. Further instructions on how to do this can be found in the CS262 Course Content folder on Blackboard.

Once you are logged into zeus, you will automatically be placed in what is called your *home directory*. This will be the directory in which you will always be placed upon logging in (unless there is an issue with zeus beyond your control). No matter what directory you might be in on zeus, you can always get back to this directory by using the *cd* (change directory) command without giving the command a directory name (i.e. just type *cd* then press Enter). At any time, if you are unsure which directory you might currently be in, you can use the *pwd* (print working directory) command. Try it now. Type *pwd* and press Enter. You should see something like:

```
/home/astudent
```

where astudent will be replaced by your username.

Commands and file/directory names on UNIX systems are case sensitive. This means that the command *pwd* is NOT the same as *PWD* or that the directory named *CS262* is NOT the same as the directory named *cs262*.

Creating Directories:

You will now create a directory to contain your CS262 labs and projects and a directory for this lab in particular. The use of directories is an easy way to keep your various assignments organized. Placing code for all your assignments in a single directory can cause significant pain and suffering, so separating each assignment using separate directories is preferred (and will generally be required as part of the submission process).

Since this may be your first time logging into zeus, you should create a general CS262 directory. The Unix command to create a directory is:

```
mkdir
```

Replace the text <directory name> with the actual name of the directory you are trying to create. In this case, you will use "CS262" (without quotes):

```
mkdir CS262
```

To ensure that the directory was created correctly, use the *ls* command (list directory contents). You should see the directory CS262 listed among other directories that are part of every user's home directory.

Now, you will change to this directory which you just created and make it your current working directory. To do this, you use the *cd* command with the name of the directory you want to change to:

```
cd CS262
```

To ensure that the command worked correctly, try using the *pwd* command again. You should see something like:

```
/home/astudent/CS262
```

Finally, you will want to make directories to keep the files and programs for whatever assignment you will be working on. **You must use a specific naming convention for files and directories.** These names will contain three things:

1. The assignment name (lab1, project2, etc.)
2. Your username (astudent in these examples.)
3. Your lab section number (201, 202, or whatever it is. Note that you will **NOT** use your lecture section.)

For example, if your username is astudent, and you are currently enrolled in lab section 204, you will create a directory for this lab (which is lab1) named *lab1_astudent_204* using the command:

```
mkdir lab1_astudent_204
```

Now, create a lab1 directory that has your username and labsection within its name. Once you have created it, *cd* to your new directory to make it your current working directory.

Creating and Compiling a Program:

Your next step is to create a hello world program in the C language. To do this, type the code below into a file named *lab1_<username>_<labsection>.c*, where *labsection* is the section number of your lab. (Again, it is suggested that you use *vi* or *vim* to create and edit this file.) Replace the text <Your Name and G Number> in the code below with your first and last name and G Number and <Lab Section Number> with the the lab section number (not your lecture section) in which you are enrolled.

```
/*
 * <Your Name and G Number>
 * CS 262, Lab Section <Lab Section Number>
 * Lab 1
 */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("Hello world!\n");
    printf("My name is <name>.\n");
    return 0;
}
```

Important Note: In general, when dealing with filenames and text within files for your assignments, whenever you see the < and > brackets, replace the text (and brackets) with the requested information. As given in the earlier example, if your email is astudent@masonlive.gmu.edu and you are enrolled in lab section 204, then you would make the filename for the code given above *lab1_astudent_204.c.* Also, the comments shown above use the /* */ style as opposed to the // style. As the semester progresses, compiler options will become stricter in regards to what is allowed in code. Although you may use the // style for now, eventually you may need to use the /* */ style, since the // comment style is not allowed in some compiler versions. Therefore, you may wish to get used to the /* */ style now.

Now that you have a program written in the C language, you will need to compile the code in order to build an executable (a program you can run from the command line) from this code. There are two ways you will do this. This first way is to compile the code directly using the *gcc* compiler, and the second way is to use the *make* command.

Compiling code using the *gcc* compiler:

Zeus has the GNU C compiler installed. It is run using the *gcc* command. There is a mountain of information available about the *gcc* command and compiler. For now, you will need to give only a few options to the compile command to create your program executables.

A general help option to the compiler is --help. Try typing the following command at the prompt on zeus:

```
gcc --help
```

You should see a list of common options that the *gcc* compiler uses. For now, the only option you will need to use is the "-o" option. This option names the output file created with the gcc command. If this option is not present, then by default, the filename of the created executable will be "a.out." However, *a.out* is not very descriptive, so it is usually best to give your program a name that is relevant to its purpose.

The next step for this assignment is to compile your *hello_world* program, and run it to verify that everything is working properly. On zeus, you will use the following command:

```
gcc -o lab1 lab1_<username>_<labsection>.c
```

The order of the parameters on the gcc command line generally do not matter. You could just as well have compiled your program using the following command:

```
gcc lab1_<username>_<labsection>.c -o lab1
```

This time, the "-o lab1" portion of the command is at the end, whereas for the earlier version, it was directly after the gcc command. However, you will need to understand that whatever name is used after the "-o" option will be the executable name. To see this, **DO NOT RUN THIS NEXT COMMAND**, but think about what will happen if you run the command:

```
gcc lab1_<username>_<labsection>.c -o
lab1_<username>_<labsection>.c
```

Look carefully at the command before reading further.

If you run the above command, you will essentially wipe out your source file. So, unless you have a backup copy, you will need to retype your entire program into a new source file. This happens to at least one student each semester, so don't let it be you! Also, consider what would be the name of the executable file if you type the following command:

```
gcc lab1_<username>_<labsection>.c
```

Do you remember what the default output file is if you don't use the "-o" option? Once again, if the "-o" option is not present, the created executable will be named "a.out."

Once your program has been compiled correctly, try to run your command. Since you used the output file option "-o lab1", you can run your program by simply typing "lab1" (without quotes) at the Unix command line. You should see output similar to:

```
Hello world!
My name is Alice Student.
```

Compiling code using the make command:

Using the gcc command is the general way to compile C programs. However, when compiling large and complex projects, it can become unwieldy. To assist compilations for such projects, a command named *make* is used on Unix systems. Although the program for this lab is simple, programs on Unix systems can become quite complex. The *make* command, in conjunction with a special file called a Makefile, is used to manage this complexity.

For this part of the lab, you will create a Makefile. You will need to create and use Makefiles for nearly all lab and project assignments in this course. For this lab, we will create a very simple Makefile. To do this, create a file named Makefile, and add the following text to it:

```
# A simple Makefile
# <Your Name and G Number>
# CS 262, Lab Section <Lab Section Number>
# Lab 1

all: lab1_<username>_<labsection>.c
	gcc -g -Wall -o lab1_<username>_<labsection>
lab1_<username>_<labsection>.c

clean:
	rm lab1_<username>_<labsection>
```

There are a few things to be sure about with the above text. As mentioned previously, items that are between the angle brackets should be replaced with your username or lab section. Also, where you see the yellow highlighting, you MUST use a tab character.

Some aspects about the Makefile will be (or have been) discussed during your lecture. Some things to keep in mind about the above Makefile are:

- The pound (#) character is used for comments. Any line that begins with a # character is ignored by the make command
- The words *all* and *clean* in the file above are called *targets*. They are used to direct the make command to execute certain portions of the Makefile
- The "-g -Wall" following the gcc command are compiler options. The -g option will add debugging information in the code (more about this in later labs), and the -Wall option will cause the compiler to report all warnings about potential issues with your code (W for warnings, all for all. So -Wall).
- The reason you need a tab for the command portions of your Makefile (gcc and rm), is part of the history of Makefiles. The original creator of the make command made a quick decision to use a tab character to help with parsing the Makefile, and after a short while, it was too late to go back to change it without breaking existing Makefiles.

Now that you have a Makefile, to compile your program using the *make* command and the Makefile, type the following at the Unix command prompt:

```
make
```

The *make* command will look for a file named Makefile, parse it, and look for the first target it finds (in this case, *all*). It will then execute the command that follows it. Since in this case, the command is a *gcc* command, your source file will be compiled using that command. If you were to type the command:

```
make clean
```

then the make command will parse the Makefile until it finds the target named *clean*, and execute the command that follows it. In this case, it will use the *rm* (remove) command to delete the executable from the directory. Because the first target in the Makefile is called *all*, using the command *"make all"* is equivalent to "*make"* (without the *all* target).

Once your program has been compiled correctly, try to run it. You do this by simply typing "*lab1_<username>_<labsection>*" (without quotes) at the Unix command line. You should see the exact same output you saw when running the version of the code compiled using the *gcc* command:

```
Hello world!
My name is Alice Student.
```

On some systems, you may need to type a "./" before the name of your executable (such as "./lab1_astudent_204"). This has to do with how PATH environments are set on the system. We will discuss PATHs and other Unix topics in later labs and lectures.

Creating a Typescript File:

A typescript file is a way to capture commands and output from a series of commands that you make at the command prompt. You will create a typescript file in your lab directory containing a listing of your code (showing that it compiles without errors) and a run of the program after compiling. To create this typescript file you will use the *script* command. This command echoes all input and output in the console window to a separate file. By default, the filename created is named "typescript." However, you will want to give the script command a specific name for your submission. Follow this procedure to produce your typescript file:

1) Create a typescript file named Lab1Typescript_<username>_<labsection>.
   To do this, type the following at the command prompt: *script Lab1Typescript_<username>_<labsection>*
   (Make sure you do not have any space characters in your filename.)

2) Show the current date and time. Type: *date*

3) Show that you are logged onto zeus. Type: *uname -a*

4) Show you are in your lab1 directory. Type: *pwd*

5) Show a listing of the current directory to ensure the source file is present. Type: *ls*

6) Show a listing of your code. Type: *cat lab1_<username>_<labsection>.c*

7) Remove any versions of lab1 that may appear. Type: *rm lab1*
lab1_<username>_<labsection>

8) Compile the code using make. Type: *make all*

9) Show that the *lab1_<username>_<labsection>* executable was created from the compile command. Type: *ls*

10) Run the code. Type: *lab1_<username>_<labsection>*

11) Now, compile the code using *gcc*. Type: *gcc -g -Wall -o lab1*
lab1_<username>_<labsection>.c
(You don't need your username handle in the lab1 executable filename)

12) Show that the lab1 file was created from the compile command. Type: *ls*

13) Run the code. Type: *lab1*

14) End the script command by typing Control-d (hold the Ctrl key and press the 'd' key).

Now you should be able to use the *cat* (concatenate and print files) command to verify that the file was created and contains the necessary information. Type "*cat Lab1Typescript_<username>_<labsection>*" (without quotes). The contents of the file will be printed to the terminal. It should contain a listing of your lab1_<username>_<labsection>.c file, the command which shows that it compiles correctly, and a sample runs of the code. Once you are sure the file is correct, go on to the next section where you will create a *tarfile* for submission.

Creating a Tarfile:

For nearly all labs and projects, you will create a *tarfile* that contains all your code and any other important files. To create one, you use the *tar* (tape archive) command. (The "tape" in the description is a holdover from earlier systems that used the *tar* command to create backups. Many systems still use this method.)

There are many uses to the *tar* command, but for this and later assignments, you will usually just archive a directory and all the files it contains. Normally, you will not include any compiled executables as part of your submission. So, unless otherwise stated by the specifications, you

will remove them using the *rm* (remove) command. You have already used this command if you followed the instructions for creating the typescript (Step 7). Be very careful when using this command! Unix systems can be very unforgiving, and if you remove a file, it is usually gone forever (unless a backup had been made before removing it). Executable programs can usually be recompiled. But if you accidentally remove a source file, you will likely have to start from scratch. So, if you haven't already, remove any executables from your current working directory.

Now that you have removed all the executables in your working directory, you can create the tarfile of your lab1 directory. To do this, you will need to *cd* to the directory above your current working directory:

```
cd ..
```

(The ".." in the command essentially means "the directory above the current working directory." As an aside, a single dot "." means "the current working directory." With this in mind, what would the command "*cd* ." do?)

Once you have changed to the directory above your lab1 directory, you should be able to do an ls command and see it listed. Also, if you do a *pwd* command, you should see "/home/astudent/CS262" as a result.

Once you are in the proper directory (the CS262 directory), you can create a tarfile containing all the files in your lab1 directory. To do this, type the following command:

```
tar cvf
lab1_<username>_<labsection>.tar  lab1_<username>_<labsection>
```

This will create your tarfile. Let's give a bit more information about the command you just used...

The "*tar*" is of course, the command for creating the tarfile. The "*cvf*" are options that you use to tell the tar command what you want to do. The "c" means "create a file," the "v" stands for "verbose" (it shows what is happening as it occurs), and the "f" stands for "filename." The "f" is particularly important because the very next word in the tar command will be the file that the tar command works on (similar to the -o option in the gcc command). So, in this case, the file you are creating will be "lab1_<username>_<labsection>.tar." Note the ".tar" extension to the file. It is very important to include this extension! Finally, the last portion of the command is the list of files and directories that will be put into the tarfile. In this case it is your lab1 directory. You may have multiple files and directories when creating tarfiles, but for most of your assignments, you will only have a single directory that you will archive.

More information on the tar command can be found by looking at the manpage for tar (type *man tar* at a Unix command prompt).

You can use several methods to ensure that your tarfile was created correctly. It is ***strongly*** suggested that you verify your tarfile before submission. Failure to do so could result in no credit to your assignment! You will learn other methods to do this in later labs, but for now, you can

use the "t" option (table of contents) for the tar command:

```
tar tvf  lab1_<username>_<labsection>.tar
```

The command should output a listing of all the files in your tarfile and look something like this:

```
drwxrwxr-x astudent/itestudent    0 2018-01-25 09:35
lab1_astudent_204/
-rw-r--r-- astudent/itestudent  217 2018-01-25 09:45
lab1_astudent_204/lab1_astudent_204.c
-rw-r--r-- astudent/itestudent 1140 2018-01-25 09:52
lab1_astudent_204/Lab1Typescript_astudent_204
-rw-r--r-- astudent/itestudent  205 2018-01-25 09:52
lab1_astudent_204/Makefile
```

File Submission:

To submit your tarfile to Blackboard, you will need to copy it to your local machine. Depending upon what type of machine you are using, there are different procedures to do this. Check the course resources on Blackboard for instructions on how to do this. Get used to doing this using multiple methods! You will be copying files to and from zeus all semester.

Once you have the file (lab1_<username>_<labsection>.tar) on your local machine, submit it to Blackboard as Lab 1.

Points to Review:

- What does the make command do?
- What is the default executable name of the gcc command?
- What does the "-o" option of the gcc command do?
- What does the "cat" command do?
- The "uname -a" command gives information about the system on which you are logged into.
- What does the "tar" command do?
- What is your "home directory?"
- What does the "current working directory" mean?
- What do the following Unix commands do?
    - cd
    - mkdir
    - pwd
    - ls
    - rm
- Are the two files Lab1.c and lab1.c the same file on zeus? Why or why not?

Get used to naming your files according to the standards given above. Submissions that do not follow the proper format may not be graded, and you will receive no credit for that

assignment.

Congratulations! You have completed Lab 1.