CS262, Lab assignment 1:
C Fundamentals and Unix Commands

Due: Sunday, Feb 13 at 11:59 pm ET

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. The instructions are mostly step by step, however, you **must not** blindly follow the procedures without understanding what you are doing. The procedures you learn in this assignment 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 programming assignments must run. You will create a directory in which to work on this assignment, 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. Nearly all future programming assignments will use these procedures, so be sure to understand **completely** what you are supposed to do.

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

- Log into `zeus.vse.gmu.edu` (or `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 all your programs work correctly on *zeus*. 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.

Information on using *vi* or *vim* can be found in the *Extra Material* pane on Blackboard. To this end, general instructions for assignments, will only be given for zeus. 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 because ALL programming assignments of this course for this semester are graded on this server.

Logging Into Zeus and Basic Unix Commands:

The first step of this assignment is to login to zeus. If you are off campus, you must use the VPN provided by GMU Tech Services, be sure to install Cisco AnyConnect Secure Mobility Client. Login 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.vse.gmu.edu
```

Replace the `<username>` in the command above with your username (email address). For example, if your email address is `lolo@gmu.edu`, then your username is `lolo`. Your password will be the same password you use for your email.

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 programming assignments and a directory for this homework in particular. The use of directories is an easy way to keep your various assignments organized. 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, 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`, etc.; Note It's NOT your lecture section)

For example, if your username is `astudent`, and your lab section number `201`, you will create a directory for this assignment named `lab1_astudent_201` using the command:

```
mkdir lab1_astudent_201
```

Now, create a directory with your **username** and **lab section** following the previous example. 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 program in the C language adding your name and lab section. To do this, type the code below into a file named `lab1_<username>_<labsection>.c`, you must use *vi* or *vim* to create and edit this file. Replace `<Your Name and G Number>`, in the code below with your first and last name and G Number and `<labsection>` with your lab section number. Give a brief description of your program. Replace `<myname>` by your first name.

```
/*
<Your Name and G Number>
CS 262, Lab section <labsection>
<lab1: Brief description of the program>
*/
#include <stdio.h>

int main(){
      char name[50] = "<myname>"
      printf("Hello world!\nMy name is %s\n",name);
      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. Also, the comments shown above use the /* */ style as opposed to the // style. 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 should use the /* */ style from 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. 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`. Type the following command 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 executable filename 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 program, and run it to verify that everything is working properly. On zeus, you will use the following command:

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

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

Note that whatever name is used after the "`-o`" option will be the executable name.

DO NOT RUN THIS NEXT COMMAND, but think about what will happen:

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

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. You can run your program by typing "lab1" (without quotes) at the Unix command line (sometimes you could need to add "./" before your exe to run it). You should see an output similar to:

```
Hello world!

My name is Juda.
```

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

```
#<Your Name and G Number>
#CS 262, Lab section <labsection>
#Lab 1

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

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

As mentioned previously, items that are between the angle brackets should be replaced with your username. Also, where you see the yellow highlighting, you MUST use a tab character.

Some aspects about the Makefile will be discussed during your lab section. 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 "-Wall" is a compiler option. The -Wall option will cause the compiler to report all warnings.
- 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.
On some systems, you may need to type a "`./`" before the name of your executable. This has to do with how PATH environments are set on the system.

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 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 *lab1Script_<username>_<labsection>*
   To do this, type at the command prompt: *script lab1Script_<username>_<labsection>*
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 homework 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 executable. Type: *rm 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. Compile the code using *gcc*.
    Type: *gcc lab1_<username>_<labsection>.c -o lab1_<username>_<labsection> -Wall*
12. Show that the *lab1_<username>_<labsection>* file was created from the compile command. Type: *ls*
13. Run the code. Type: *lab1_<username>_<labsection>*
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* command to verify that the file was created and contains the necessary information. Type "*cat lab1Script_<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.)

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. Be very careful when using this command! Unix systems can be very unforgiving, and if you remove a file, it is usually gone forever. Executable programs can usually be recompiled. But **if you accidentally remove a source file, you will likely have to start from scratch**.

Now that you have removed all the executables in your working directory, you can create the tarfile of your directory. 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.". Once you have changed to the directory above, 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 homework directory. To do this, type the following command:

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

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.

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.

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 and after submission. Failure to do so could result in no credit to your assignment! 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 2022-01-31 09:35 lab1_astudent_201/
-rw-r--r-- astudent/itestudent  217 2022-01-31 09:45 lab1_astudent_201/lab1_astudent_201.c
-rw-r--r-- astudent/itestudent 1140 2022-01-31 09:52 lab1_astudent_201/lab1Script_astudent_201
-rw-r--r-- astudent/itestudent  205 2022-01-31 09:52 lab1_astudent_201/Makefile
```

File Submission:

To submit your tarfile to Blackboard, you will need to copy it to your local machine.

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

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?

Congratulations! You have completed your first assignment.

Errata:

[Jan, 31] Page 6/8 Step 11 must be:

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

[Feb, 1] Page 8/8

… submit it to Blackboard as `lab1_<username>_<labsection>`**`.tar`**