Lab 1: Your first C program(s)

Due date: Tuesday, October 1, beginning of the lab period.

Lab Assignment

Assignment Preparation

Lab type. This is an **individual lab**. Each student will submit his/her set of deliverables.

Collaboration. Students are allowed to consult their peers[1] in completing the lab. Any other collaboration activities will violate the *non-collaboration* agreement.

Purpose. You will encounter a number of C programs, will learn how to compile, run, edit and create C programs and continue your acquaintance with the CSL Linux environment. You will get acquainted with the testing framework that we will be using in the upcoming labs. You will also download files from the course web page and learn how to view Postscript and PDF files on CSL machines.

Preliminaries. You are assumed to have successfully completed the setup instructions from Lab 0. In particular, I am assuming that (a) you can log into your CSL account; (b) you can open multiple terminal/xterm windows; (c) you can navigate the directory structure in your home directory; (d) you can create and edit text files using your preferred text editor.

The Task

Note: Please consult the instructor if any of the steps are unclear, or if the reality does not match the instructions below.

1. Set up a directory for Lab 1. You should have the `cpe101` working directory created in the home directory of your CSL account from Lab 0. Change to this directory. Create a directory for Lab 1. Change to the newly created directory. You will be putting all files you use in this lab in it.

2. Get the files! Your next task is to put a few files into your Lab 1 directory. Perform the following tasks.

- Start a web browser. Firefox is the most popular browser in the Linux world. You can start it by typing

[1] A peer for the purpose of CPE 101 is defined as *"student taking the same section of CPE 101"*.

```
> firefox&
```

in the terminal window (*notice the "&", it will let you continue using your terminal window for further commands!*), by clicking on the web browser icon on the system menu, or by selecting Applications ⟶ Internet ⟶ Firefox Web Browser from the system menu.

Note, Google's Chrome browser may be available on some machines, but due to temporary lapse in Google's support for Chrome for our specific flavor of Linux (called CentOS), you are advised to use Firefox. Chrome installs, where they exist are out of date.

- Open the web page for the course. The url is

  http://www.csc.calpoly.edu/∼dekhtyar/101-Fall2013/

- On the course web page locate the Labs section. Find "Lab 1" line in the table. You should see links to the Postscript and the PDF versions of this lab handout (this document), and a third link, "Data and Tests". Follow the latter link. A new page, containing the list of files necessary for Lab 1 will open.

- From this page (http://www.csc.calpoly.edu/∼dekhtyar/101-Winter2012/labs/lab1.html), you need to download the following files into your Lab 1 directory:

  - `hello.c`: a simple Hello, World! C program.
  - `magic.c`: a C program illustrating the use of our test framework.
  - `checkit.h`: a C file containing testing macros (read below).

  Download the files by opening them in the browser and then selecting the File⟶ Save Page As option from the browser's menu bar. Make certain the files are saved in your lab 1 directory.

- You need to get one more file, but this time, you will be using Linux `cp` command to copy the file to your Lab 1 directory. The file name is `aboutme.alex` and the file is located in the ∼`dekhtyar/www` directory. Use the following command to copy it:

  ```
  > cp ~dekhtyar/www/aboutme.alex .
  ```

  Note: "." in this command refers to current directory. **MAKE SURE YOU INCLUDE IT IN THE COMMAND!** The command reads *"Copy `aboutme.alex` file from the ∼`dekhtyar/www/` directory into the current directory.*

- Type `ls` to confirm that all four files are now located in the your Lab 1 directory.

- Let us now make sure you can peruse both PDF and Postscript files. In your browser, return back to the main course page. Find the handout for Lab 1 on the course web page. You will be offered two download formats: PDF and Postscript. Select download in PDF format. Select `"Open with ...Document Viewer"` if prompted. The application that opens your PDF file is called `evince`.

- Now, download the file in Postscript format. In the dialog window, select "Save to disk" and save the file to your `cpe101/Lab1` directory.

- Use the `ls` command to make sure that your directory contains the Postscript file you downloaded. Type

```
> gv <FileName> &
```

replacing `<FileName>` with the name of the downloaded PostScript file (it will have the extension `.ps`). The program that opens the file, `gv`, of `GhostView` is a PostScript file viewer. Make sure you can navigate the document (move from one page to another), and study other features of `gv` that are immediately available to you. Exit `gv`.

- Now, type

```
> evince <FileName> &
```

again, replacing `<FileName>` with the name of the downloaded PostScript file. Note, that `evince` can be used to view PostScript files as well as PDF files.

You are now ready to view/peruse all course materials.

3. Compile and run the programs. Time to put the files you downloaded to use.

- Let's view the contents of each file. The command that prints the contents of a file to a terminal is `more`. The format of the command is

```
more <filename>
```

where `<filename>` is the name of the file you want to view. Using the `more` command, output the contents of `hello.c` and then - `magic.c` to the terminal.

You can also output the contents of `checkit.h` to the terminal, but the C code in that file may be rather hard to read/understand.

- Study the contents of the `hello.c` file. This is the famous Hello, World! program written in C.

- Let us compile this program. You will be using GNU C++ compiler `gcc` to compile your C programs in this course. `gcc` recognizes pure C programs and compiles them accordingly. Type

```
> gcc hello.c
```

- Nothing happened. You get a new Linux command-line prompt. What gives? Well, let's see if anything has changed in your Lab 1 directory. Type

```
> ls -al
```

If you did everything correctly, the output will look (roughly) as follows:

```
total 19
drwx------  2 dekhtyar nobody    6 Sep 26 13:43 .
drwx------ 16 dekhtyar nobody  135 Sep 26 13:34 ..
-rwx------  1 dekhtyar nobody 6425 Sep 26 13:43 a.out
-rw-------  1 dekhtyar nobody 1502 Sep 26 13:21 checkit.h
-rw-------  1 dekhtyar nobody  364 Sep 26 13:25 hello.c
-rw-------  1 dekhtyar nobody 2353 Sep 26 13:25 magic.c
```

There is a new file, `a.out`, which was not there before. And (if your terminal if set up right), this file name will be `green`. This is the result of compiling `hello.c` file. (It is green because this is how Linux terminal marks executable files).

- Let us run this program. Type

```
> a.out
```

What's the result? Congratulations, you just ran your first C program (albeit, you did not write it).

Note. If the above command does not produce expected results (and you get a `"a.out: command not found"` error message, try running `a.out` as follows:

```
> ./a.out
```

Use the same trick (`./` in front of the program name) for all other commands to run executable files produced by the C compiler in this lab.

- Let us now examine the second program. Use `more` command to view its text again. The `magic.c` file contains a lot of comments that try to provide brief explanations of each line of code. Read these comments carefully.

- We will be compiling `magic.c` using a more complex command. In particular, this command will ensure that `gcc` does all of the following things:

  - uses ANSI C standard for syntax checking (`-ansi` flag);
  - reports all compilation errors and warnings (`-Wall` flag);
  - compiles using *strict* rules - all warnings are treated as errors (`-Werror` flag);
  - tells `gcc` to actually correctly process some mathematics (`-lm` flag);
  - changes the name of the output executable file from the default `a.out` (`-o` option);

  Type

```
> gcc -ansi -Wall -Werror -lm -o magic magic.c
> ls
```

  If you typed the `gcc` command correctly, you now should have an executable file `magic` in your directory.

- Run the `magic` program. The output you should see is

```
Test passed on line 34.
Test passed on line 35.
Test passed on line 36.
Test FAILED on line 41.  7 is 7, expected 8.
Test FAILED on line 42.  11 is 11, expected 8.
```

We will discuss what this output means in Task 6.

Note: For most, if not on all labs and programs, we will be using the `gcc -ansi -Wall -Werror -lm` command to compile your submissions. This will force you to write code in pure ANSI C, and will force you to identify, debug and remove all and any compile-time errors and warnings.

5. Create your first program. Since you already have the Hello, World! program, the first program you create will be more complex.

- You will be modifying the `hello.c` program. First make a copy of it, that you will be using. Using the `cp` command make a copy of `hello.c`. Name your new file `aboutme.c`.
- Edit `aboutme.c` file using a text editor you are comfortable with. You have to change the contents of the `aboutme.c` in the following manner:

  1. All lines in a C program that start with `"/*"` and end with `"*/"`. These lines are called *comments*. C compiler ignores all text in these lines. This text is for the programmers to specify what program this is, who created it, etc. . . .

     Modify comments found in `aboutme.c` to include the following:

     – mention that Alexander Dekhtyar is the instructor of CPE 101.
     – the section you are in.
     – your name.
     – change the title of the program to indicate that this is an `"About me"` program.

     Note: All changes above are to be done ONLY in the comment lines.

  2. Now, edit some code. Examine the code of the current program. It should be clear by now, that the actual printing of the `"Hello, World!"` text is accomplished by the

     ```
     printf("Hello, World!\n");
     ```

     line. Note the following about this line:

     – C programs consist of statements. `printf("Hello, World!\n")` is a statement of the C program.
     – `printf()` is a *special* C function whose purpose is to output information to screen. While we will learn much more about `printf()` in near future, for now, we use its simplest version as above: the text to be put on screen is provided in the parentheses.
     – The line ends with a semicolumn (";"). In C, semicolumns are used to separate different statements. When you edit your program below, each `printf` statement must end with a semicolumn. Forgetting to include one for each statement is the single most common compilation error you will encounter during this course.
     – The text to be printed, `Hello, World!` appears in double quotation marks ("). C uses double quotes to represent `strings`, i.e., textual information the program deals with.
     – The text inside the double quotation marks ends with `"\n` but the output (as observed in previous runs) does not contain `"\n"`. What is it? To determine what `"\n"` does, delete it from the line, (leave the line to be `printf("Hello, World!");` save the program, compile it (use

```
> gcc -ansi -Wall -Werror -lm -o aboutme aboutme.c
```

command), and run it. What is the difference between running `aboutme` and `a.out` program that you got by compiling `hello.c`? Can you figure out what `"\n"` means and does?

3. Restore the `printf` statement to its original form. Add another `printf` statement immediately below the first one. Type

```
printf("\n My name is <Name>.\n");
```

replacing `<Name>` with your name (in my case, it'd be `Alexander Dekhtyar`). Save the program, compile and run it. What do you see now?

4. Edit the program to add more `printf` statements. The text in the statements should contain some brief information about you - your major at Cal Poly, when you joined Cal Poly, what you like to do, etc.

   A short version of such text for your isntructor is available in the `aboutme.alex` file (remember — you copied it some time ago). This file contains the output of the instructor's version of the `aboutme.c` program.

   Please note, that the text you put should be readable (i.e., well-formatted, including line breaks in reasonable places), and should be truthful. I may use the outputs generated by your program in determining group and pair designations for future labs and/or program assignments.

6. Get acquainted with our testing framework. In our study of programming and software development, *testing* will play an important role.

We define **testing** as *the set of activities a software developer performs in order to ensure that the program (s)he is working on performs as expected and produces correct results.*

In the first half of the course, we will use the `checkit.h` framework to test your code. `checkit.h` (developed by Dr. Aaron Keen for use in CPE 101) is a collection of C *preprocessor directives* (otherwise called *macros*) that can be used in C programs to test how parts of the program perform.

`magic.c` uses one directive defined in `checkit.h`. The directive is `checkit_int()`. This directive compares the values of two *integer C expressions* and reports whether the values supplied in the code were the same (successful test) or different (failed test). `magic.c` uses `checkit_int()` five times. The first three times yield success while the last two times result in failed tests. The following expressions are compared:

Test	First Expression	Second Expression	Result
Test 1	10	10	`Success`
Test 2	10	5+5	`Success`
Test 3	5+5	20-10	`Success`
Test 4	7	8	`Failure`
Test 5	11	4+4	`Failure`

Your assignment is as follows. In addition to the five tests above, `magic.c` contains ten more tests (tests #1 through #10) in comments. Each test is incomplete. The first expression for each test is provided, while the second expression is replaced with the `<?>` placeholder. You need to:

1. Uncomment all `checkit_int()` tests in the program.

2. For each uncommented `checkit_int()` test, substitute the `<?>` placeholder with the **C constant** which makes the test **succeed**.

This is best achieved in a step-by-step fashion. Uncomment the first test. Replace the placeholder with the constant integer value that makes the test succeed. Compile the program (using the `gcc -ansi -Wall -Werror -lm -o magic magic.c` command). Run the program. If the test fails, replace the incorrect constant with the correct one. If the test succeeds, proceed to uncomment the next `checkit_int()` test.

Note, that if you try to compile the program with a `checkit_int()` test, in which the `<?>` placeholder is not replaced, the compiler will report an error.

Example. Consider the following `checkit_int()` test:

```
/* checkit\_int(1+2+3, <?>); */
```

The first expression provided to the test is 1+2+3. We can compute the value of this expression: $1+2+3=6$. We uncomment the test and replace the `<?>` placeholder with the computed value 6:

```
checkit\_int(1+2+3, 6);
```

After this, we compile the program and run it to ensure that the test succeeded.

Note. `checkit_int()` test contains `printf()` statements which output whether the test was successful or failed. The printed message reports the line number in the C program from which the test originated. This allows you to uniquely determine which tests are being reported, which specific tests succeeded and which - failed.

7. Submit your programs. You are at the finishing line for this lab assignment. All that is left is to submit your work. You will be submitting two files, your `aboutme.c` program and your `magic.c` program modified as specified above.

In all your coursework, you will be using a program called `handin` to submit your work. The typical submission format for `handin` is

```
> handin <instructor> <assignment> <fileName1> <fileName2> ...
```

Here,

- `<instructor>` is the CSL LoginId of your course instructor. My loginId is `dekhtyar`.
- `<assignment>` is the designation of the assignment, for which you are submitting. Current lab assignment designation is `lab01`.
- `<fileName1>, <filename2, ...` are the names of all the files you wish to submit. For this lab you are submitting just one file, `aboutme.c`.

To submit, first `ssh` to `unix1`, `unix2`, `unix3` or `unix4`, or `vogon`, then navigate to the Lab 1 directory and issue the following `handin` command:

```
> handin dekhtyar lab01 aboutme.c magic.c
```

(your entire session should look similar to this (your directory paths may be different):

```
> ssh unix1
Password:
unix1> cd cpe101/lab01
unix1> handin dekhtyar lab01 aboutme.c magic.c
unix1> exit;
>
```

Note: You can resubmit your files as many times as you want until the deadline. Past the deadline, submission will be kept open for 24 hours to collect any late submissions (you can submit as many times in that period, but you run the risk of earning progressively larger late penalties), after which, the submission will be closed. Once the submission is closed, all subsequent attempts to use `handin` for the particular assignment will be rejected.

Good Luck!

Appendix C: Working on CSL machines remotely.

Due to limited time the labs will be open in the upcoming quarter, you may have to work on some course assignments from your home machine and/or laptop on various occasions. Fortunately, if you have a desktop/laptop computer, you can connect remotely to CSL machines. All programs that you write in CPE 101 will use only a single terminal window for input/output, and therefore, working via remote connection is a feasible way to complete all assignments. Additionally, you can set up an environment that would allow you to do your work on your machine and simply use remote connection for submission of completed work.

Note: Regardless of where you complete your work, please note that I will be grading your code on CSL machines. **It is your responsibility to ensure that the code you submit runs properly on CSL machines.**

MS Windows Computers

If your computer runs a version of MS Windows OS (XP, Vista, Windows 7, etc), you can download a few small programs to establish remote connection to CSL machines. You can also install a well-known Linux emulator which includes `gcc` compiler.

Remote connectivity. You need two programs: an ssh client for connecting to the CSL workstations and a file transfer client for transferring files between local and remote machines.

- SSH client. SSH, a.k.a., Secure Shell Protocol is the means of information exchange between computers, which allow one computer to remotely access another. Download and set up a Windows SSH client `PuTTY`, which is found here:

  http://www.chiark.greenend.org.uk/~sgtatham/putty/

  Upon download of the executable (`putty.exe`) place it on the desktop and start the program. In the window that opens do the following:

  1. enter one of the following CSL server names to connect to[2]:

     ```
     unix1.csc.calpoly.edu
     unix2.csc.calpoly.edu
     unix3.csc.calpoly.edu
     unix4.csc.calpoly.edu
     vogon.csc.calpoly.edu
     multicore.csc.calpoly.edu
     ```

  2. Select "SSH" radio button.
  3. Open the connection. You will be prompted with the information about the publich key signature of the server you are connecting to and asked if you want to proceed. Answer "Yes". A terminal window will open, and will prompt for user's loginId (enter your CSL loginID) and password (enter your password). If successful, you will see the CSL Linux prompt appear.

[2]Due to maintenance, not all of these servers may be available all the time. If you are trying to connect to one of these and have no luck, try another.

- File Transfer Client. I use `WinSCP`. It can be downloaded from here:

  http://winscp.net/eng/download.php

  You can download either the installation package or a single portable executable.

Local Environment. The best way to set up your work under a Windows OS is to download `Cygwin` - a Linux emulator for Windows. You need to download and install both `Cygwin` and the `XWindows` emulator. `Cygwin` can be obtained from:

http://www.cygwin.com/

Download and run `Cygwin`'s `setup.exe` file (make sure you have Internet connection when you run it). `setup.exe` will allow you to select what you want to install (for simplicity, you can choose the default packages — it is sufficient for our purposes), and will allow you to install the XWindows server as part of the overall installation process.

Linux

If your machine runs one of the multitude of Linux flavors, then, you can access CSL machines remotely using the following command:

```
> ssh <loginID>@<server>.csc.calpoly.edu
```

Here `<loginID>` is your CLS login Id and `<server>` is the name of one of the CSL servers (see list above).

Also, all Linux installations come with `gcc` compiler installed. So, you can use your favorite text editor and `gcc` to reproduce the lab environment.

MacOS X

If you have an Apple machine running MacOS X, you essentially have a copy of a Linux alternative called `FreeBSD` running.

Remote connection. MacOS X comes with its own SSH client. The instructions on how to set it up can be found at

http://www.panix.com/help/sw.macosx-ssh.html.

Local environment. MacOS X has its own terminal emulator, which you can use to enter command from command line. `gcc` compiler is usually not installed on MacOS X systems, however, it can be downloaded from

http://connect.apple.com/

See

http://developer.apple.com/tools/gcc_overview.html

for the overview of `gcc`. Registration is required, but the download is free.