Mechanics of Programming CSCI.243

Homework 1: System Introduction

Due Date: 6/13/2017 at 11:59pm

1. Overview

In this assignment you will practice using the CS department's Ubuntu® machines. These machines are the required hosts for all electronic submissions of homeworks and projects. You will need to use a terminal window and UNIX®/ Linux® programming tools to get materials, edit code and text files, and submit your work. In addition, you will need to use the CS department's programming tools, `get`, `try`, and eventually `gmakemake` to do your assignments.

2. Resources

2.1. UNIX/Linux

You need to become familiar with the basics of Linux (and, by extension, other UNIX-like systems) to do your assignments this term. Unless you are already very experienced with Linux, you should complete at least parts 1 and 2 of the http://www.ee.surrey.ac.uk/Teaching/Unix/ tutorial.

2.2. Editors

On our Ubuntu Linux systems, there are several editors available, including:

```
pico nano gedit nedit vi vim
```

The `gedit` and `nedit` are graphical, while the others work inside a terminal window.

We recommend learning `vi` or `vim` due to their extensive use in the UNIX/Linux community. The `emacs` editor is another powerful, versatile editor, although fewer faculty and students in the department are experts.

The http://www.openvim.com/tutorial.html covers the basics of `vi` and `vim`. You also can take the http://vim-adventures.com tutorial that teaches the editor using an adventure game.

2.3. SSH: Secure Shell for Remote Access

When you cannot come to the CS labs to work, you need to use a *secure shell (SSH) client* to connect from a remote system such as your own computer to a CS department Ubuntu machine with the correct configuration.

You need to do a *remote login* from your machine and use your CS account, not your institute DCE account to connect. When logged in through ssh, you will have to use a plain text editor that has no graphical interface unless you learned to run the X11 Window System on your local computer and enabled X11 within putty/ssh.

Host Machines for Using SSH

To get a list of Ubuntu systems for remote login, run this command in a terminal:

```
cat /etc/hosts | grep 129.21.37 | grep Ultra | grep i386
```

and choose a host name from the list. This command lists machines in ICL1, ICL2, and ICL3, and the ones with the longest names tend to have fewer users on them.

SSH Clients

On your local machine you need to (install and) run a secure shell (SSH) client program. This is specific to the operating system on your machine.

- Putty for Windows SSH

  Download and install the PuTTY executable. When you run putty, enter the host information, connect, and then enter your CS account and password.

- Linux SSH

  Linux users will be already familiar with the terminal. Using your *CS-account-name* and chosen CS *machine-name*, run the following command and then enter your password when prompted.

  ```
  ssh CS-account-name@machine-name.cs.rit.edu
  ```

  For options that you may find handy when initiating ssh sessions see the program's manual page using `man ssh`.

- Mac OSX SSH

  Run the Terminal application program found in the Applications/Utilities folder. Running ssh on a Mac is the same as in Linux.

3. Account Setup

It is time to create a useful directory structure for storing your assignments. You should not use the "WIMP" (window, icon, mouse, pointer) techniques with which you may be most familiar. Instead, you should navigate, create folders, list them and determine your working directory using these shell commands inside a terminal window:

```
        cd mkdir ls pwd
```

Create the following directory folder structure under the Courses subdirectory of your CS account's HOME folder. Plain text and indentation below indicates directory hierarchy. This should be the output of the `tree` program after you `cd Courses` and enter the command shown here:

```
$ tree ./Courses/CS243
./Courses/CS243
|-- Homeworks
|   |-- 1
|   |   `-- your-hw1-files-go-here
|   `-- 2
|       `-- your-hw2-files-go-here
`-- Projects
    `-- 1
        `-- your-project1-files-go-here

5 directories, 3 files
```

For each subsequent assignment, you will `mkdir` the appropriate folder in the correct place and use that as the location for your work on that assignment.

3.1. Registration for `try`

The course uses a locally maintained program named `try` for submission and evaluation of student work.

`try` limits who is allowed to submit work. The registration process gets you set up in the `try` database for the course.

The try registration command is:

```
        try grd-243 register /dev/null
```

If you make a mistake when registering, please contact your instructor as soon as possible. Until your registration is correctly entered, you will not be able to submit assignments. The instructor will be able to fix it.

The Student Guide to Try will help you get the most out of `try`. It is also important that you read the Try Summary to understand how to use the system and not get tied up by it.

3.2. Fetch the Assignment with `get`

Using the cd command, change to the folder for this assignment. Retrieve the materials by typing this:

```
get csci243 hw1
```

This will copy the files for this assignment from the course account to your current, working directory.

---

4. Assignment Tasks

This section describes multiple activities to complete for credit on this assignment. Always do the activities in the order in which they are listed.

4.1. (30 pts) Studying Erroneous Code

For this activity you need:

- `questions1.txt`: A file containing questions to answer; and
- `warning.c`: A complete C program to fix.

The standard command and command line arguments you should normally use to compile your code in this course are:

```
gcc -std=c99 -Wall -Wextra -pedantic
```

The names of the C source file(s) you wish to compile must appear at the end of this command.

If you are compiling to create a `.o` file and not compiling to create an executable file, you must add this option flag to the list:

```
-c
```

If you are compiling to create an executable file, you should use this flag and filename sequence:

```
-o filename
```

That will specify the file name that will contain the executable. (Future lectures will cover these options in more detail.)

After you figure out the problems with the program, fix them, and edit the `questions1.txt` file to add your answers to the questions.

Submission

Submit your solution using the following command:

```
        try grd-243 hw1-1 questions1.txt warning.c
```

Submitting with `try`

Some rules and guidelines for using `try` are found under the course's Resources page: Try Summary

4.2. (30 pts) Linking Compilation Units, Header Files

In this activity you need:

- `questions2.txt`: A file containing more questions to answer;
- `main.c, circle.h, circle.c`: A complete program that has compilation problems to diagnose, explain and fix.

Read the code carefully. Try to compile it. After you have figured out the problems with the program, fix them, and edit the `questions2.txt` file by entering the answers to the questions.

Submission

Submit your solution using the following command:

```
        try grd-243 hw1-2 questions2.txt circle.h circle.c main.c
```

4.3. (10 pts) Understanding diff output

When source files are submitted using try it is normal for the try system to build them into an executable file and run tests. The output of your program is compared against the output of the solution. This comparison is performed by a utility called `diff`, which finds differences between the two inputs. Some of the tests performed by try have visible results that are displayed to you. This lets you see what your program is doing wrong and gives you clues that may help you correct it and submit the work again. A good understanding of the diff output is needed so that you can interpret it correctly.

In this section you will experiment with diff and learn to interpret the output. First create a new file called test1.txt, containing these 3 lines of text:

```
abc
Hello World
Hello World2
```

Create a second new file called test2.txt, containing these 3 lines of text:

```
Hello World
Hello World2
```

```
Test
```

Use the `diff` program with these command line option flags `-bcw` to compare the two files:

```
        diff -bcw test1.txt test2.txt
```

The following example demonstrates the symbols used to indicate differences that involve entire lines.

The output should be similar to this:

```
*** test1.txt   Sun Jan 24 21:24:31 2016
--- test2.txt   Sun Jan 24 21:24:44 2016
***************
*** 1,3 ****
- abc
  Hello World
  Hello World2
--- 1,3 ----
  Hello World
  Hello World2
+ Test
```

The first 2 lines show that the "***" symbols represent the first file, and the "---" symbols represent the second file.

The line that contains "*** 1,3 ****" indicates that the next lines after it are lines 1 through 3 of the file represented by the *** symbols, i.e. the first file.

The line that contains "--- 1,3 ----" indicates that next lines after it are lines 1 through 3 of the second file.

The `diff` program is comparing these two snippets against each other. The symbols at the start of each line indicate what type of difference exists, if any.

Notice the *minus sign* in front of the "abc" line. That indicates that the entire line "abc" is missing from the second file. The *plus sign* in front of the "Test" line indicates that this line has been added to the second file; i.e. it is entirely missing from the first file. The lines with nothing (except blank spaces) in front of them are identical.

Now consider an example where there are differences within a line. Modify the test2.txt file so that it contains 4 lines composed of:

```
Hello World
heLLo World2

Test
```

Diff the files again using:

```
        diff -bcw test1.txt test2.txt
```

The output should be similar to:

```
*** test1.txt   Mon Jan 25 09:39:37 2016
--- test2.txt   Mon Jan 25 09:40:30 2016
***************
*** 1,3 ****
- abc
  Hello World
! Hello World2
--- 1,4 ----
  Hello World
! heLLo World2
!
! Test
```

Notice the `!` symbol at the start of several lines. This symbol is used to indicate that a difference exists. Note that it appears in front of "Hello World2" in the first file and in front of "heLLo World2" in the second file. This indicates that there is a difference within this line. Additionally there are ! symbols in front of the blank line and the "Test" line in the second file. These lines are completely missing from the first file. The diff program chose to indicate that a difference exists by using the ! symbol. In the previous example we saw that a `plus sign` indicated the existence of an entirely new line in the second file. As the diff utilty parses through the second file, it may choose to indicate the existence of an extra line using a **plus** sign, or it may identify it as a difference and use an `!` symbol.

Now consider a more complicated example where the files contain both lines that are identical and lines that are different. Modify test1.txt so that it contains 12 lines:

```
abc
Hello World
Hello World2
1
2
3
4
abcdefg
123456789
Hi
Goodbye
Testing
```

Modify test2.txt so that it contains 10 lines:

```
Hello World
heLLo World2
1
2
3
4
abcdefg
123456789
Hi
Goodbye
```

Again diff the two files. The output should be similar to:

```
*** test1.txt   Mon Jan 25 21:59:13 2016
--- test2.txt   Mon Jan 25 21:59:04 2016
***************
*** 1,6 ****
- abc
  Hello World
! Hello World2
  1
  2
  3
--- 1,5 ----
  Hello World
! heLLo World2
  1
  2
  3
***************
*** 9,12 ****
  123456789
  Hi
  Goodbye
- Testing
--- 8,10 ----
```

This output shows that there are two areas within the files that have differences. Each one begins with a line of "*************", followed by a snippet from the first file and then a snippet from the second file. The first comparison shows lines 1 through 6 of test1.txt and lines 1 through 5 of test2.txt. The second comparison shows lines 9 through 12 of test1.txt were compared with lines 8 through 10 of test2.txt. In this second comparison the only difference is that "Testing" does not exist in test2.txt, so diff shows this by using the - sign. The other 3 lines, "123456789", "Hi", "Goodbye" are shown for context to make it easier to know what is being compared even though they are identical between the two files. Since they are identical diff can indicate this just by displaying "--- 8,10 ----" for the second file.

The `diff` Activity

For this activity you need to open myfile.txt and examine its content. Then submit it to try using:

```
        try grd-243 hw1-3 myfile.txt
```

Try will compare your file to the solution file and display a diff. You need to interpret the diff output and modify myfile.txt. Continue to submit it until it matches the solution. When it matches the diff utility produces no output.

4.4. (30 pts) Implementing a Simple Algorithm in C

In this activity you need to complete:

- triangle.c: The skeleton of a program that prints text-based triangles of various sizes.

Complete the function that prints triangles. The triangles should be made of *asterisks*, and the program output should look like the following:

```
*

  *
 ***
*****

   *
  ***
 *****
*******
```

If the function's argument is even, the function increments the size by 1. The main program should print the 3 triangles one right after and the other.

Your program must use loops; you *may not hardcode* the statements to print each of the specified triangles, and you may not use recursion.

Submission

Submit your solution using the following command:

```
        try grd-243 hw1-4 triangle.c
```

4.5. Activity Summary

- 30 points: erroneous code
- 30 points: fixing, compiling and linking a multi-file program
- 10 points: learning `diff`
- 30 points: writing a simple program

Update history:
2017/01/18 wording updates
2016/08/25 UNIX/linux/etc. updates
2016/08/21 original version
