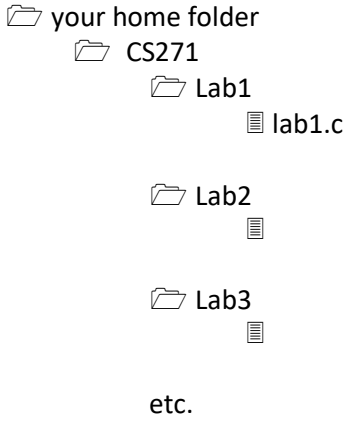
CS 271
Lab Assignment # 2

Submit **calls.c** and **triples.c** on Canvas.

Read Chapter 3 and 4.

1. Log in to your Computer Science Linux account. Organize your directories (folders) and files so that you have something like this:



Here are some Linux commands that may be helpful:

Command	**What it does**
`cd` *`directory_name`*	changes working directory to *directory_name*
`cd ..`	changes working directory to one level up
`cd home`	changes working directory to your home directory
`pwd`	displays the working directory
`mkdir` *`directory_name`*	makes a new directory
`mv` *`source destination`*	moves (or renames) the source to the destination The source may be a file or a directory.

2. In the terminal window: Remove the object files (files that end with .o). Carefully type the following commands:

cd	(press Enter...takes you to your Home folder)
cd CS271/lab1	(press Enter...takes you to the CS271 lab 1 folder, assuming that you named the folders correctly.)

*** from here on I'm going to assume that you know to press Enter after a command ***

rm *.o	Caution: rm deletes things, there is no undo, and no "are you sure?" message

Note: If you put a space in a filename or folder name, you'll need to surround it with quotes when you type it in a Linux command. For example: cd "lab 1"

First Program: calls.c

3. Download the file data2.txt from Canvas and save it in the Lab2 folder you created.

4. Change working directory to the Lab2 folder. Create a program named **calls.c**.

Problem: A technical support center tracks the lengths of calls that come in. At the end of each day, a report must be created containing summary statistics: total number of calls, mean call length, minimum call length, and maximum call length.

Input: A text file containing one line for each call. Each data value represents the duration of a call (in seconds). The sentinel value -1 indicates the end of input data.

Input terminates when the value -1 is entered for call length (use a sentinel loop).

Processing: Count the number of calls. Don't include the -1 at the end. Calculate the mean call length (you'll need the count and the sum). Keep track of the minimum call length and the maximum call length.

Output: Align the output as shown below. (The numbers shown here are not the actual values. They're just to show how the output is to be formatted.)

Total Number of Calls	100
Mean Call Length	58.6 seconds
Minimum Call Length	12 seconds
Maximum Call Length	1240 seconds

***Do not use arrays. There is a penalty in the rubric.
***Do not write functions other than main. There is a penalty in the rubric.

Style Requirements

- When you write an inline comment, place it on a separate line before the program statements that perform the task. Use only the single line comment syntax. (Yes, there is a penalty in the rubric for using multiline comments.)

- Include header comments in the following format. This header format is expected on all programs you submit this semester.

  // CS 271 - lab assignment #
  //program_name
  // purpose of the program
  // written by *yourname*
  // date written

5. Save the program. Compile

```
gcc calls.c -o calls
```

6. Debug if needed and recompile. Run and test the program.

```
./calls
```

   Enter a few numbers. When you run the program this way, it will get all input from the keyboard (you have to type the data values, then type -1 at the end of the data).

   Make sure that you calculated the expected output and compare to the actual output. Perform at least 3 tests with different numbers to make sure that your program is working correctly.

7. **Input Redirection** Run the program again using input redirection. This will take all input from the file you specify (instead of the keyboard).

```
./calls < data2.txt
```

   View the contents of the data2.txt file in your editor and calculate the expected output of the program. Compare your hand calculation to the actual program output. If the values don't match, debug your program or correct your hand calculations (or both).

   Close the program and data file once you have the correct output.

Second Program: triples.c

8. Create a program named **triples.c**.

   "C How to Program", 8th edition, problem 4.27 on page 154. In other editions, it may be on a different page.

   Here's the problem... A right triangle can have sides that are all integers. The set of 3 integers for the sides of a right triangle is called a Pythagorean triple.

   Write a C program to find all of the triples with hypotenuse less than or equal to 1000.

   Use brute force with triple-nested for loops.

   - Include documentation just like you did in calls.c.
   - In the output, print column headings, neatly aligned columns, and right-justify the numbers as shown below:

```
Side 1     Side 2     Hypotenuse
------     ------     ----------
     3          4              5
     5         12             13
...
```

9. Compile. Debug if needed and recompile. Run and test the program. You can check your output for this one by doing a web search on “Pythagorean Triples”.

Submit calls.c and triples.c on Canvas.