Lab 3

In this and subsequent labs you will start to create slightly more complex programs. For each lab you will be given a series of steps to complete the work. As a bit of self encouragement, you will be awarding yourself "accomplishment points" for completing steps which practice or introduce a new concept. For steps which are somewhat, or even considerably (for later labs), more involved the self awards are increased. Please try to not go more than 4 steps ahead without having your work checked by the GA.

Importantly, you need not bother to read through this (and the subsequent labs) **carefully** (although do scan through the labs well prior to lab day) unless you are doing that while actually seated at a computer as you proceed... for some of the labs there are quite a number of steps and you will have time to do these during the lab. However, if you try to follow all the steps in detail just in your head, you'll likely get a bit lost and it may not be time well spent (for which there are no accomplishment points).

On the other hand, for some labs you might want to think about how to do certain portions when scanning through - for example, in this lab you may wish to give step 6 some preliminary thought prior to the lab period.

Note that while many do, not all steps necessarily build upon each other to produce your finished coding. You are writing code in some of the early parts (especially in the later labs) to ensure that you can correctly generate code to perform certain tasks later in that lab's work. The capabilities this elementary work develops will be needed to generate the finished program. [An easy way to not have to retype things is to comment out code you don't want to run at the moment (with /* ... */), and copy it into the running part of your program as you need it. Code not needed in the end should be removed just before submission.]

Run each little task to check that your code works before claiming the award for that level.

1) Using Quincy, write a C program that will use a **prompt** (it will print out a line to ask the user) to input 4 numbers into 4 **integer** variables (choose sensible names for these, eg **num1**, **num2**, etc). Give yourself 1 accomplishment point.

2) Next have your program produce and output the **sum** of those four numbers. Be sure the output has a **label** which tells the user what the output value represents. (eg: **Sum of the four values is: 23**) Give yourself 1 accomplishment point.

Always **label** output in this manner and be sure to **prompt** for all input values (unless explicitly told not to do so - when dealing with input from files later in the course, for example, this will change).

3) Next have your program output the **sum** of the **first two numbers** minus the **sum** of the **last two**. As this is not a huge step forward, give yourself just 0.5 accomplishment points.

4) Next have your program produce and output the **sum** of the **squares** of the four numbers. For the same reason give yourself only another 0.5 accomplishment points.

5) Next have your program produce and output the exact **quotient** (a real number with a decimal point - a **double**) of the **square root** of the **sum of the squares of the numbers**, divided by the **sum** of all the numbers. eg for input values 3 4 5 6 this is 0.51 [square root of (9+16+25+36) divided by 18]. Show the output with two decimal digits of accuracy.

Note that with the input values as integers, before performing the division you will first have ensure it contains at least one double. You can convert the **sum** to a double by casting, or divide by 4.0 as opposed to 4, or by multiplying the sum (or the integer divisor, 4) by 1.0 (a double) prior to the division taking place. Otherwise the average can be off a bit.

Try producing the incorrect average by not performing any conversion to doubles (if the average happens to come out correctly without the conversion, change one of the input values so that it doesn't); then produce the correct exact average by properly converting the one (or more) of the terms involved in the quotient to a double.

Have the output shown with four decimal digits of accuracy.

Give yourself 1.5 accomplishment points.

6) Write another C program on paper (do not use the computer for this step) that creates a table of distance equivalents in yards and miles for 100 m, 200 m, 400 m, and 800 m. You would input each value into a variable and then convert it first to yards and then to miles by multiplying it by the appropriate conversion factor. A meter is equivalent to 1.094 yards and 0.0006215 miles.

Your program should right justify all the numbers (see your **printf** notes from the lectures), aligning them nicely in a column (table) form.

Have the GA briefly inspect your program before starting to write it on the computer. Once accepted, give yourself 1 accomplishment point.

7) Now enter your C program into Quincy, and run it. You could encounter two aspects where corrections need to be made. First, it is possible that the quick GA check might not catch all of any logic errors you might have created (programming which simply does not make sense or does not properly cover the requirements the program is supposed to address); such errors are much easier to spot once the code is actually in the machine. Second, although your coding on paper might be fine, you may have introduced syntax errors as you typed in the program.

Removing both kinds of errors is known as "debugging", and you will almost certainly need to "debug" your program. Programs without bugs are rarely, if ever, produced as a first attempt - and certainly not once a certain level of complexity is present.

Submission: Submit both your programs as completed as per the lab instructions