CECS 174 – Assignment 8

Part A – Do-While Loop – Newton's Method of Finding a Square Root – Given a function *f(x)*, in this case a square root of some positive value *v*, and an initial guess at the solution x_0, you can iteratively get a very close approximation to the square root of a number using Newton's method.

Newton's formula for finding a root of a function is: $x_{n+1} = x_n - (f(x_n) / f'(x_n))$

In our case *f(x)* is $\sqrt{v}$ which is the same as $x^2 = v$. Which means $f(x) = x^2 - v$.

Which also means that the derivative $f'(x) = 2x$.

Now we can plug those into Newton's formula: $x_1 = x_0 - ((x_0^2 - v) / (2*x_0))$.

As we said before, *v* is the value that we are trying to find the root of and x_0 is our initial guess at the solution. The better the guess, the faster the calculation will converge to the solution. A solution to a square root will be a number that is greater than zero and less than *v* (since it must be the positive root), so you will want your guess to be somewhere within that region.

Once you have plugged your guess into the formula, you will get a result for x_1. Compare the values of x_1 and x_0. If they are very close together then it means that your solution has converged to an approximate solution, if not, then you need to try another iteration of the calculation. Your new number, x_1, is now a good number to use as your guess, so replace the value of x_0 with x_1 and do the calculation again.

Repeat this process using a do-while loop until your solution has at least 5 decimal places of accuracy, and then print the solution to the screen. Also display the result of the Math library's sqrt() function for comparison. Check the user's input for validity and repeat the program until the user enters the sentinel value 'Q' to quit.

Before you begin coding, work out the problem by hand until you understand the process. Then draw a flowchart to diagram Newton's method (ie. not the error checking and sentinel value parts).

Example Output:

```
Enter a value: 50
Newton's Square root: 7.0710678118654755
Math Lib Square root: 7.0710678118654755
```

Part B – For Loop – Computing Primes – Prompt the user for an input (*n*) to calculate up to. Find the prime numbers up to *n* using nested for loops. Prime numbers are those that cannot be evenly divided by any other numbers except itself and 1. Display 5 primes per line and check the user's input to make sure it is an integer and non-negative. (Hint: 1 is not a prime number, but 2 is, so start your counting from 2).

Example Output:

```
What number should the primes be calculated up to? 50
2 3 5 7 11
13 17 19 23 29
31 37 41 43 47
```