**EECS 297/397 Special Topics: C/C++ Programming** Spring 2020

# Assignment 1: C++ Basics, Variables and Control

Instructor: Orhan Özgüner Due: February 06 before 11:59 PM

This assignment consists of three required components and one bonus.

## Part 1

You must first modify the Babylonian square-root estimation algorithm discussed in class. At the moment, the algorithm runs for a finite number of iterations and stops. This means that for a number like, say, 3, where the initial guess starts out close to the actual answer (1.5 vs 1.732...) the accuracy will be extremely good. However, for much larger numbers where this is not the case, the accuracy will become disproportionately poorer.

You will need to create a version of the algorithm which takes in two numerical inputs- $S$ is the number to take the square root of, and the other is an $\varepsilon$ value which determines how accurately you want to approximate $\sqrt{S}$. Instead of running for a finite number of iterations, the algorithm should run for as long as it takes for the change to the estimate to fall below $\varepsilon$, and only then output the result.

## Part 2

This part involves implementing a *new* square-root computation algorithm, sometimes called Newton's Method (based on the Babylonian method). This method finds the reciprocal square root of $S$ (that is, $\frac{1}{\sqrt{S}}$), which can then be used to find the desired $\sqrt{S}$ with the multiplication

$$\sqrt{S} = S \cdot \frac{1}{\sqrt{S}} \tag{1}$$

```
procedure NEWTON($S$)
1   $guess \leftarrow \frac{2}{S}$
2   while condition
3       $a \leftarrow \frac{guess}{2}$
4       $b \leftarrow 3 - S \cdot guess^2$
5       $guess \leftarrow a \cdot b$
6   return $S \cdot guess$
```

This method can also theoretically run forever, continuing to refine its estimate, so you should also modify it to ask the user for a $\varepsilon$ value and run conditionally on the update step changing *guess* by more than $\varepsilon$ (that is, stopping when the update step changes *guess* by *less* than $\varepsilon$).

## Part 3

Finally, you may notice that for numbers where the initial guess is sufficiently far away from the actual square root (particularly very small numbers, like 0.01), the algorithm in Part 2 never converges but in fact diverges- the changes in *guess* get bigger and bigger, and move it farther and farther away from $\frac{1}{\sqrt{S}}$. Implement logic to detect this (we will say that the algorithm has started to diverge if the difference between guesses increases for two consecutive iterations- in fact there are cases where the difference will increase and then settle down again, but we don't need to worry about those right now), stop iterating, and print out an error message.

## Part 4 (Bonus)

So far, we have been assuming that the user will only enter numbers into the solver. This is fine for a proof of concept, but production code should not assume that users are always polite enough to do what they're told! Modify one of the above 3 parts to check the user's input. Ask the user to enter a number or the letter 'q' to quit. If 'q' is entered, the program should exit immediately. If a number is entered, the solver should continue as before. If something other than just 'q' or a valid number is entered, the solver should produce an error message and ask for the same input *again*. This should continue for as long as the user enters invalid inputs.