Alex Hoffer

Assignment 4

1. Program Design:

a) The question wants me to calculate the square root of a number in two ways: using the Babylonian algorithm, and using the math library square root function. The first way is where most work comes in. I am to determine which way I want to use the Babylonian algorithm: that is, do I involve the user? How much? I have decided to use the algorithm twice: once by using an initial guess from the user and also a loop count provided by the user, so that we can get a good feel about how the algorithm approximates the right answer. The second way is I will use the initial guess of n/2 and count how many loops it takes for the algorithm to reach the value calculated by the math library. I should do plenty of printing to make sure the user knows what is going on within the algorithm. The main thing I am expected to know to complete this problem is how to implement the Babylonian algorithm. Luckily, it is written out in the details, and so I am only responsible for the context in which it is used.

b) I will modularize the problem using several functions. These functions are main, getInput, userBabylon, and programmerBabylon. I will start at main and call getInput, where I will get the number that the user wants the square root of. Then, I will pass that number to userBabylon. This function will ask the user to provide an initial guess about what the square root of that number is. It will also ask the user for how many loops through the algorithm they'd like to perform. At the end of the function, the algorithm will be worked through to the correct number of loops, and the value it gets as the square root will be returned to main. Then, for comparison, I will call the function programmerBabylon, passing the original number to it. This function will assume an original guess of n/2, as suggested by the assignment. It will use a while loop with the condition that the number to be used to calculate the square root using the algorithm is not equal to the actual number, provided by the math library. The number of loops it must go through to reach this value is recorded, and returned to main. Finally, the number provided by the math library is printed out.

c) Test Plan:

Value	Expected	Actual
Math function calculates square root correctly. Sqrt(4)	2.	2.
Enter -5 as number to start with.	If statement converts the -5 to positive 5, prints out warning to user.	Same as expected.
Enter 4.5 as number to start with.	The int portion of the code translates 4.5 into an integer.	Same.
Enter 0 as number to start with.	If statement handles it, prints out warning, re-calls function.	Same.
Programmer Babylon function should be able to get to the same answer as the math function. Ask for it to find the square root of 4 with an initial guess of 2.	The function will print out that it required 1 loop to reach the same number as the math function.	Same.

2. Computational Thinking:

Computational thinking is the devising of methods to efficiently and accurately solve problems involving numbers. The “involving numbers” part of that sounds like it limits its scope, but this is not the case when one realizes every natural phenomenon can be represented with numbers (whether a given representation does the phenomenon justice is beyond the general scope of these short essay questions). This sort of thinking is crucial to computer science because programs are written on the same basis: a program should solve a series of problems accurately and efficiently. The ability to computationally derive solutions to problems is part of computer science, and the other part is the ability to notice ways in which these derivations can be sleeker, so the program can run faster and more smoothly. I think the statement is a bit redundant and creates a dichotomy I don’t think is effective. She says computer science is more than programming a computer, and then proceeds to describe the skills necessary (“thinking on multiple levels of abstraction”) to indeed program a computer. I believe her point is that computer science requires and breeds computational ability as well as intuition and imagination. You must be well-versed in how numbers interact with each other, but you also must be able to imagine new ways in which these interactions could be used to produce new programs. I have found that computer science requires a unique blend of math and common sense. You must understand how to program, but you also must think about how you’re thinking and approaching a problem when you’re doing it. That way, you can make your methods as obvious as possible to you in the future or somebody who is combing through your code. It is valuable for all people to learn computational thinking because it introduces a pragmatic edge to the way you approach problems. It breeds a stepwise way of living. You become more aware of what you’re doing and how you could do it better. It helps you break down big problems into small modularized chunks which can be approached one at a time. In this class, I have learned how to transfer C++ ability to Python, which has been a testament to the versatility of computational thinking. That is, I think computational thinking involves the notion that learning how to do things one way opens your eyes about similarities in doing things in a different way, and enables a transfer in knowledge.