Project #: ________2
Semester: ___Spring
Name: Sam DeCook

I. Requirements: Restate the problem specification and any detailed requirements in your own words.

Create a class that implements the use of complex numbers. Most functions are already defined for you, but you must create a constructor that receives a C-style string as it's parameter, a multiplication overload, and a istream overload.

II. Design: How did you attack the problem? What choices did you make in your design, and why? Show class diagrams for more complex designs.

For the constructor, I went through the logic considering what would be required if there were two, one, or zero numbers found using `sscanf`. I made use of the existence, and or the position of the character "i" when making the if – else statements.

The multiplication function was easy, I made use of the definition of complex multiplication.

The istream function was also easy, as I was able to take advantage of the C-string constructor I had just created.

III. Security Analysis: State the potential security vulnerabilities of your design. How could these vulnerabilities be exploited by an adversary? What would be the impact if the vulnerability was exploited?

The input isn't scrubbed, so if you knew the language it was written in, you could inject code into the program. Depending on the level of access the user has, they could inflict considerable damage.

IV. Implementation: Outline any interesting implementation details in your solution.

Using `sscanf` was a lifesaver, it helped me efficiently parse the C-string.

I also made use of a `const char*` to the location of the character "i" using `strchr`. If there was only one double parsed, the existence of the "i" character informed me whether the double was a real or imaginary number. I was also able to use it to check for an implicit "i" if the character just before the "i" was a "+" or "-".

V. Testing: Explain how you tested your program, enumerating the tests if possible. Explain why your test set was sufficient to believe that the software is working properly, i.e., what were the range of errors for which you were testing.

This assignment was made significantly easier from the fact that allowable input was defined for us, and we didn't have to take into account ill-formed input. That meant that I was able to just trace the logic through and implement it into if – else blocks. Once the tests on ZyBook came back good, I was confidant my solution was correct.

VI. Summary/Conclusion: Present your results. Did it work properly? Are there any limitations? NOTE: If it is an analysis-type project, this section may be significantly longer than for a simple implementation-type project.

It does work properly. You could say that a limitation exists because we can't accept ill-formed input, but that's outside the scope of this project.

VII. Lessons Learned: List any lessons learned. For example, what might you have done differently if you were going to solve this problem again?

Make sure I was using the `sscanf` function instead of just the `scanf` one. Other than that, I learned about the importance of tracing logic through all of its possibilities. I also learned to not test your code code after every change you make, rather spending time to make sure that the logic is correct and you haven't made any silly errors.