Mohammed Azad
CISC 3160 : Lab 5:

Note:
The old code and the rewrite are in the github page provided, this is a reflection based on those two programs.

Looking back, it is very easy to tell that I did not have a knack at writing good code. Although this is still true to this day, it seems that it was much worse when I first started. Almost every rule that I would consider today to be a part of good coding standards seemed to be sidelined and all of this contributed to me reading old code that took more time to understand than it did to rewrite it in a better way. At first glance, what I noticed was how short the code was and it gave a sense of relief thinking that this would not take too long to understand. However, that first glance was probably the highlight of the experience, as the actual reading of the code was dreadful. It didn't help that I totally forgot what the objective of this code was for and the sole criteria for picking this program was due to it being saved on my desktop folder for the past few semesters. What contributed to the sorrow that I felt when reading this code was the lack of proper structure, the very obscure names of the variables, and what felt to be a lack of making a better effort for maintainability. However, looking closer at the code I noticed that the output was looking to either recall or not recall something, there were numbers in a specific range, and there was an input string being compared with the values of "chevrolet" and "oldsmobile" and by using these clues I was able to piece together what the overall objective of this code was. The objective was to take an input of a car's model name and model year number and to check if the car was either a chevrolet or oldsmobile and if the car's model year number was of a specific year range. Should the comparison match, then the console should output that this car should be recalled otherwise it should not be recalled.

Now that figuring out the purpose of the code was complete, it was time to decide how to fix the codebase. Spending the past few semesters heavily steeped in the computer science department makes the experience of rewriting this code much easier. However, knowing what I know now, such as very theoretical aspects of computer science, I know what aspects of the code should be focused on. A notion that I tried to incorporate with the new version of the code is to try to apply the Unix philosophy, where many small parts that do things well are pieced together to make a whole program work properly. I noticed that in the original code, it was trying to do everything at the same time and it made the code unaesthetic and difficult to understand. The new version, although it took more lines of code to write, is more structured and properly explains what is happening. This is further helped by using variable names that are meaningful and manipulating boolean values in a way that increases maintainability and increases utility by compounding them. It should be noted that the new version of the code is written in Java for no special reason other than the original version was written in Java. Also the objective did not have problems that required special characteristics that a certain programming language would cater to, and as such it would be fairly arbitrary to pick a programming language to solve this problem. Viewing the code now, it seems that I was too obsessed with taking the “*end justifies the means* “ approach and this contributed to poorly written code and, although it solved the problem, it was purposely limited by how it was written. To some degree, I still take this approach, however it is mitigated by the amount of standards that I have to persevere learning to make writing the code easier in the long run and it involves thorough planning and time management.The newer version properly assigns tasks and values to the correct places and is able to bring them together to make more concise and understandable code. And again, the problem being tackled was fairly simple and would not require pseudocode necessarily nor would it require an extreme amount of planning. In this case the best method, for me, was to first create small parts and glue them together, and if that didn’t work I would figure why it didn’t work and what could make it work. This is all a result of experience with writing code and I think that was the most lacking feature of my previous self. It is very easy for me to criticise the code that was written by a younger me, however there is a sentimental aspect to it that I did not expect. A sort of reminiscing in how these simple problems used to feel so hard when doing them, but now I can breeze past them, and I think that is best categorized as growth.