MAL Project 1: Syntax Compiler

Total Time Accumulated: 10 hours

This project had a profuse number of ups and downs along the way. Whenever I “thought” I had the correct, methodical, way to writing my code, it would break, and I would have to find a new method. I actually learned a lot with some of C#’s libraries which came in full affect when I was stripping the program from it’s blank lines and comments. I decided to create an ErrorLabels class which holds the error’s description and also it’s title so that I can access both when calling only one object. This method worked out perfectly in the end when I had to write the final report at the footer.

My initial schedule was a decent start, but I had to scrap a lot of my plans when trying to strip the program from its blank lines and comments. I ended up using LINQ which uses slightly different syntax for operations. For example, LINQ uses lambda expressions which is new to me.

Going through each line of code wasn’t too bad. It required some fine looping and some rigorous concentration. So, I used an abundant amount of ‘foreach’ loops to cycle through each line that was read from the file. Then, inside the foreach loop, I created a string array that stripped the line into sections so that I would have, {“LoadI , R4 , =file”}. Then I called a method inside of the foreach loop (after I stripped the lines), and I analyzed each element in the array and checked for blatant errors. So, in my opinion, my approach was very efficient, but also confusing if a random programmer was trying to go through it. Luckily I only used around 200 lines of code not counting my ErrorLabels class.

I only used 3 errors for my program and I only caught 2 of them. I didn’t use the Ill Formed Identifier and a couple others because I wanted to keep this program as elementary as can be. I did not want to make the volume 1 edition of this program too complicated so that later on in the future (If I return to this program), I can just add on to it.

The main thing that I saw was that MAL does not use warnings at all. With modern day IDE’s, we are always reminded with small warnings that could exacerbate if we are not careful. Higher-level language IDE’s also have recommended variable name spelling. Which is nice. I honestly cannot think of any more discrepancies between the errors of assembly language and the errors of high-level languages. They both have out of bound exceptions but are printed out differently, they both have invalid opcode type exceptions but with different ways of explaining it to the programmer. I still do think that now with higher-level programming we have more tools and assets to assist us with impeccable programming. Warning labels always come in handy when programming because they can exacerbate if we are too ignore it (not all the time). It also prefers us to use a more conventional way of naming our variables compared to methods and classes so that we have a more universal way of writing code. This helps readers and other programmer who need to access your code.