Saad Teeti

New York University in Abu Dhabi

In this project I have developed a program that takes an input file text file of arithmetic equations, solves it, and output it on the screen and also outputs it in a text file if the user chooses to. In order to make this code I created multiple functions, and throughout the process of making this program I had to delete and edit some of the functions for my program to come out the way it is now. This report includes the flow of the program and some challenges that faced though out the process.

Description of the flow in the Program:

The most difficult function was the Tokenization. Before I started tokenization, I pushed each line from the text file into vector eq, so I can perform tokenization. For tokenization, I separated each equation based on three types. The first type was the variables, the second type was operators such as +,-,*, and the third type was Numbers. We check for some character as they will always be single character in the tokens and if they are there then we push them. After pushing them we move to the character after, if the character after is an Operator of +, -, *,/, then we push each type on its own so we can find the unary operations. For example, multiplication, we check the first character and then check the second one if they are the same then we push it into the same vector. If not the same, then each one would be by its own token. But if we have more than 2 characters of the same operator followed by each other, then we push them and preform tokenization on them in another function called fix.

In fix function we do a new tokenization, we check if their number of operators are even then we token each two together, and if its odd then we start by tokening each two together from the right until we reach to the left and we would have one at the left. So for example if we have 3 minuces, it would be tokenized like [+][++]. Then we converted the tokens to {"!" plus the unary token operator.} Then I made an Infex to Postfix function and in order to implement it, I used stacks.

After converting from postfix to infix, I did the evaluate function. For the evaluation function, I made a Boolean function that first check if the equation is invalid, then it won't calculate it, and if the equation was calculated before then not to calculate it again. Then checks if the token is a variable and if it is valid, and also checks if the tokens is a number. Then using stacks, I apply operators and I push the result to the stack. Then I go case by case and evaluate each operation, and then push the result. Then I was able to use C++ maps for the results.

Challenges:

A challenge that I faced was dealing with unary operators. When I started my code, I came up with a function unary to binary in which I convert each unary operations into a binary ones. I was able to realize that this way might have not been effective, because it would be really hard to hard code every single case possible for all of the unary operators, therefor I came up with the idea of add ! before every unary token then evaluating it at the end in the evaluation function.