CS21120 Supplementary Assignment Report, 2020-2021

Task 1: The Virtual Machine

For this task I started from the beginning and worked my way through the document trying to understand both subtasks before tackling the actual problem. Once I grasped the idea of what I was creating I set up my IDE and started working on the instructions first. I worked through each class in the order they came up within the document so I wrote the easier instructions and then the more challenging ones. The directions in the assignment helped me grasp the idea of what I had to accomplish in each a lot more than I thought they would. In order to really understand how to write the code in this section I took a step back and created the class methods first, wrote some basic comments so I could see what the important keywords were and then got to work. As for the more complicated subtract and divide instructions assigning variables really helped me understand just how to make these two classes work; obviously I could not test the work I had done just yet as my virtual machine was not yet implemented.
As for the virtual machine I got stuck trying to understand what to do here for a little while as my knowledge of implementation, so for this I took a step back and tried to understand what to do with a bit of common English; I knew I needed a private field stack with the "Double" type due to get result being a double so this part was easy for to grasp the idea of. After this I tampered around with the "Execute" method within the "VirtualMachine" trying to cast objects to doubles, strings to doubles and not really understanding what was going wrong. Then after looking through the interfaces and understanding I needed to use the execute stack from within the "IInstructions" interface I managed to understand the concept and used the private stack within a for each loop within the execute method to pass a reference straight to the stack; I then created a result variable within "getResult" and ran my tests and found they all passed without any problems.
The Big-O complexity of the execute method within "VirtualMachine" I believe would be Linear Time O(n) this is due to the fact I am using a for loop so the more data input in the loop the more time the algorithm will need to run.

Task 2: The Parser

In order to write this code I first created the class and added the interface "IParser" and also added the "SyntaxException" class. I then read through the task 2 brief in order to understand how the parser works and made some comments within the "parseExpression" method as this algorithm seemed very complex looking at it from the beginning. I started working on what I did know; creating the stack and queue instances of output and operator, creating the outline of the "if-elseif-else" statement and throwing a new "SyntaxException". Then I started looking at the tokenizer and working out how that worked which wasn't too difficult so I assigned the output of the token to a string and starting working on how to check if the string represents a number; I figured out that I could create a method that returns true if the input (tok) would be a double and false if anything else. The hardest part of this method was figuring out how to check the precedence of the operators inside the stack and within the new instruction. This confused me for quite some time until I refreshed my memory on accessing interface methods and when I figured this out I finally managed to finish the while loop condition; finally, I added the tests to my program and all of them passed (excluding bracket tests). I believe the maximum number of items in parsing is 7 and in the execution stack it is 3, I used .size() to check the sizes when running tests on the items. A virtual machine

would not be needed for a simple calculator as there would not be any huge numbers being used or to the power of expressions etc. To get rid of the virtual machine you could take the members of the virtual machine which would be the stack, execute() and getResult() members and move them into a different class from the virtual machine and implement this class into the parser.

Task 3: Brackets

This section of the code was quite easy for me to work through as the challenging parts of the algorithm to tackle were in task 2. The hardest part of this section was figuring out how "instanceof" worked within the program; I was struggling as I had been creating new instructions each time in while loops which would not work with "instanceof". The way I tackled it was to separate the instance of section of the while loop which allowed me to just call "BracketInstruction" without a new instance of it. I don't believe the maximum size of each changed as even with brackets within the method both parsing and virtual machines sizes stayed the size.

Task 4: What problems remain?

There are a few mathematical expression that cannot be parsed by this algorithm which the most common would be modular "%" as it is not handled in the method and also raising numbers to a power is also not able to be done as the expression "^" can also not be handled within the method. A problem with this parser is that it cannot handle multiplying negative numbers as it treats "-"as an operator and not a negative number in a sum. The parser also cannot handle other types of bracket such as [] as these are not present within the method. It also fails to handle roots like square root and also factorials. A further problem would be not being able to calculate trigonometry functions such as sine and cos.

Self-Evaluation:.

I found that the parseExpression method within the parser was somewhat challenging to code when looking at it from the beginning; but as I worked through it I found that the more I coded the easier it was to grasp the concept of the algorithm, in terms of answering some of the questions within this report I got quite confused with the Big O complexity and really need to look back over notes to calculate this. I believe I did most of the coding sections very well especially adding the brackets into the algorithm after finally finishing the original parser. I believe the mark I will get will be quite high as most of my code works well and passes all the tests within the package provided but I may lose a few marks inside this report due to the questions I answered not being accurate or incorrect.