**Declaration on Plagiarism**

**Assignment Submission Form**

This form must be filled in and completed by the student(s) submitting an assignment

|  |
| --- |
| Name(s): Toba Toki |
| Programme: CASE4 |
| Module Code: CA4003 |
| Assignment Title: Cal Assignment 2 |
| Submission Date: 12/12/2018 |
| Module Coordinator: David Sinclair |

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I/We have read and understood the Assignment Regulations. I/We have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

I/We have read and understood the referencing guidelines found at

<http://www.dcu.ie/info/regulations/plagiarism.shtml>, <https://www4.dcu.ie/students/az/plagiarism>and/or recommended in the assignment guidelines.

Name(s): \_\_\_\_\_\_\_Toba Toki\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_16/12/18\_\_\_\_\_\_\_\_\_\_\_\_

**Semantic Analysis**

# Abstract Syntax Tree & Symbol Table

Creating the AST was a systematic process. I had to decorate my production rules for the ones that were needed for semantic analysis. The purpose of the AST was to remove away and information that is deemed as redundant meaning that the alteration was a necessary one.

For the symbol table class, I included Hashtables. The job of these Hashtables was to map the scope keys to a linked list of all the variables that would be defined in that scope. It was also to map a key that is the addition of the scope and an id, to a data type. The Hashtable also maps the same key that it encounters to a category description such as “variable”, “constant” etc. The symbol table by default is initialised with a global scope that is mapped to an empty list.

There are a lot of methods in the symbol table class. These methods range from inserting an entry into the symbol table to checking for matching variables in the same scope. There is also a print symbol table method which outputs the symbol table to standard output. There is also a method that returns true if a variable id is in the supplied scope. It also consists of methods that return the data type associated with the id and scope that is already supplied.

# Semantic Analysis

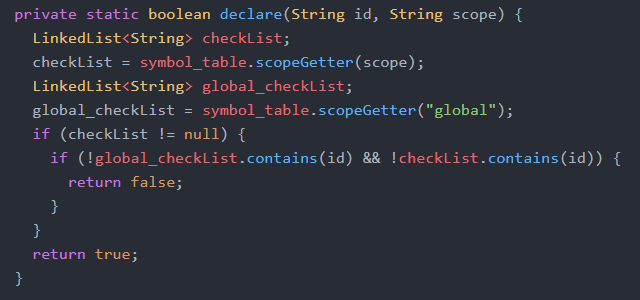
For this segment, I made “SemanticAnalyser” which implements “calVisitor”. The reason for this was so that each node would be visited recursively so that the semantic checks could be done. The default scope from the beginning was set to “global” but this would change depending on where the scope moves to. If the scope moved from global to function, this scope would change. A number of semantic checks were listed out to be completed.

* Is every identifier declared within scope before its is used?

In order to check if a variable had been declared before it could be used, I implemented a Boolean function that would check the current scope and the global scope to see if they contain the declaration.

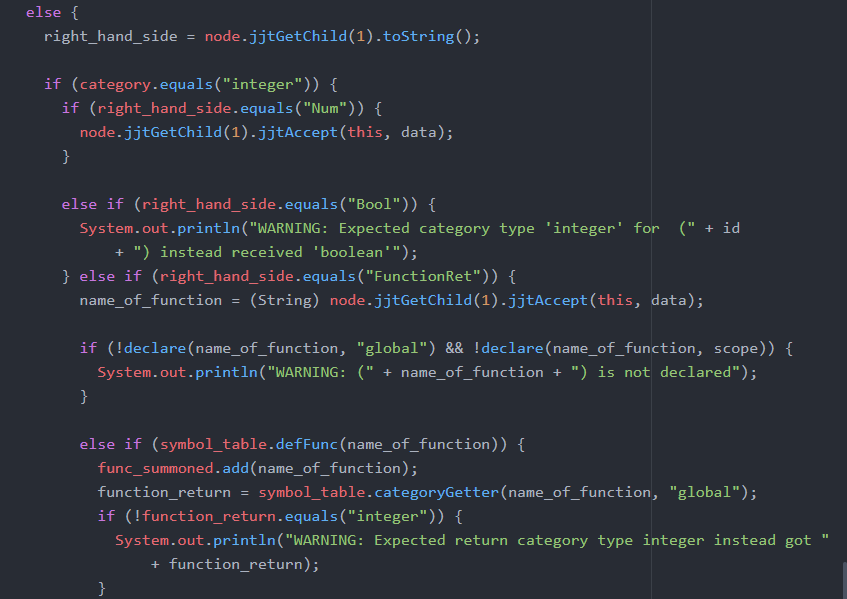
* Is no identifier declared more than once in the same scope?

In order to perform this check, I had to make a no matching function that checks to see if a same variable is declared more than once in the same scope. It keeps track and stores every time a variable or a constant is entered. It also checks to the global scope to see for multiple declarations. The LinkedHashSet will avoid storing duplicate values.



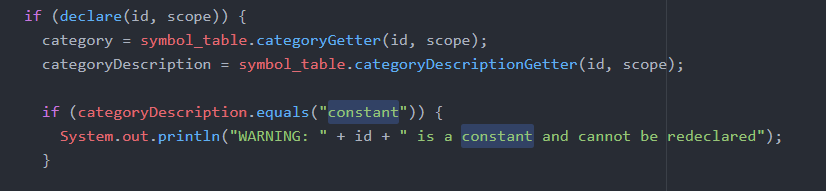
* Is the left-hand side of an assignment a variable of the correct type?

In order to perform this semantic check, I split the statement node into a left hand side as well as a right hand side. For every id, I check the symbol table to see what category the id should be. If it is an integer then I search for Num node. I search for the appropriate node for the left hand side.



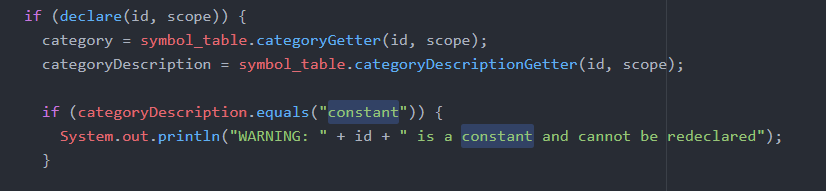
* Are the arguments of an arithmetic operator the integer variables or integer constants?

I checked to see if a variable is being declared to see if it is a constant, or if it is that an error as constants cannot be redeclared. An error message will be shown which will prevent any operations with the left hand side.



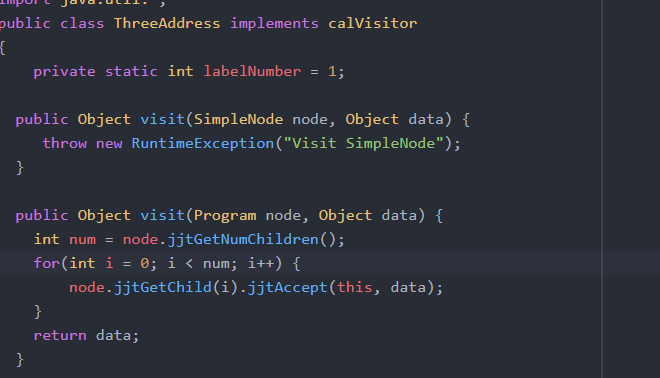
* Are the arguments of a boolean operator boolean variables or boolean constants?

I checked to see if a variable is being declared to see if it is a boolean, or if it is that an error as Booleans cannot be redeclared. An error message will be shown which will prevent any operations with the left hand side.



## IR

IR code is generated by my ThreeAddress class. It visits each of the nodes recursively. The class provides two helper functions. One for printing instructions and one for printing labels. This is done so that the code is printed in the right and correct format.



## How to run:

Jjtree cal.jjt

Javacc cal.jj

Javac \*.java

Java cal test1.txt

To run against another test file, simply change the name.