**CA4003 Assignment 2**

**Name(s):** Samuel Wood

**Programme:** Computer Applications, Year 4

**Module Code:** CA4003

**Assignment Title:** Semantic Analysis checks and intermediate representation for for the CCAL Language

**Submission Date:**  2019

**Module Coordinator:** David Sinclair

**Student Number:** 16476006

I understand that the University regards breaches of academic integrity and plagiarism as grave and serious. I have read and understood the DCU Academic Integrity and Plagiarism Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach this policy. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the sources cited are identified in the assignment references. I declare that this material, which I now submit for assessment, is entirely my 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 work. By signing this form or by submitting this material online I confirm that this assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. By signing this form or by submitting material for assessment online

I confirm that I have read and understood DCU Academic Integrity and Plagiarism Policy (available at: http://www.dcu.ie/registry/examinations/index.shtml)

**Introduction**

The aim of this assignment is to add semantic analysis checks and intermediate representation generation to the lexical and syntax analyzer you have implemented for the CCAL language from assignment 1. To complete this assignment the following tasks must of been completed

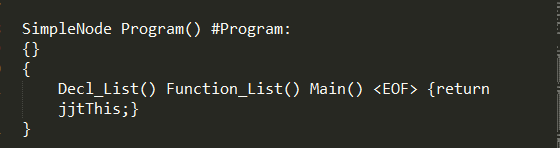
* Generate an Abstract Syntax Tree
* Add a symbol table that can handle scope
* Perform a set of semantic checks.
* Generate an Intermediate Representation using 3-address code

**Abstract Syntax tree**

The first step of this assignment was to develop an Abstract Syntax Tree. This meant making use of the JJ file from the previous assignment and using JJtree, the preprocessor for Javacc that is used to build the abstract syntax trees. I had to take the contents of the JJ file and place it in a jjt file of the same name and running ‘jjtree CCAL.jjt’ on the file. This generates the new jj file and other java files that are required. Some options in the JJTree file also had to be set to true.

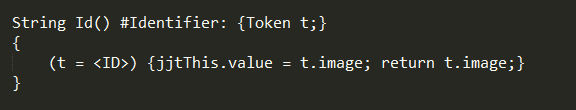
‘MULTI’ was set to true so a decoration, for example ‘#EXP’ would generate a node derived from the class ‘ASTEXP’. This is an example of one of the many files that are generated when the jjtree command is run. VISITOR is also set to true so a jjtAccept method is put in each node class and to generate a Visitor interface.

My Program method from assignment 1 had to be slightly modified to a SimpleNode instance so the abstract syntax tree could be implemented. This was the change to the program() function.

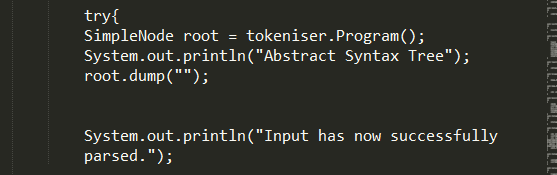


If you look closely there is a hashtag placed after the definition. As stated above this generates a Node derived from the corresponding AST class. I did the same for other functions I felt were suitable in my grammar to see where they appear in the tree.

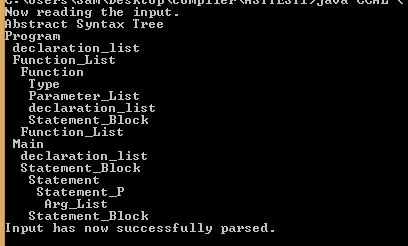
Also, so the Abstract Syntax tree could be properly implemented I had to add an element to the grammar and modify others. One was ‘String Id’ so the ID token could be returned. I also had to alter the Type method in the grammar to ‘String’ instead of ‘void’ so it could also be returned. Here is the new string element of the grammar below.



After further research in the course notes and of the SimpleNode file and its “dump” function. I thought to implement the print statement when the SimpleNode or Program method is called. This was done before the ‘Input has now successfully parsed.’ message as you can see below.

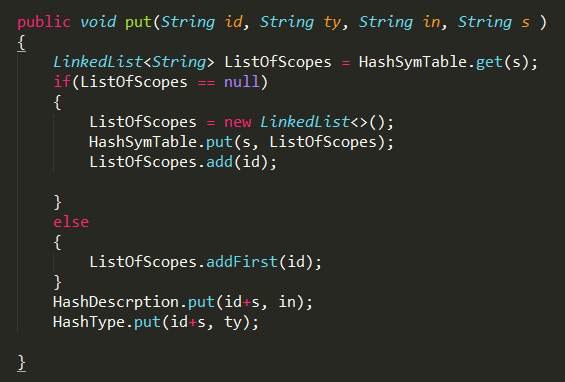


Below you can see the Abstract syntax tree printed for the second example test case I had saved from the previous assignment.



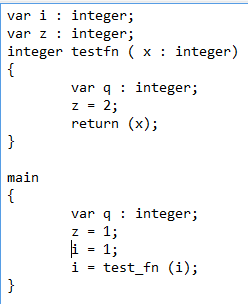
**Symbol Table**

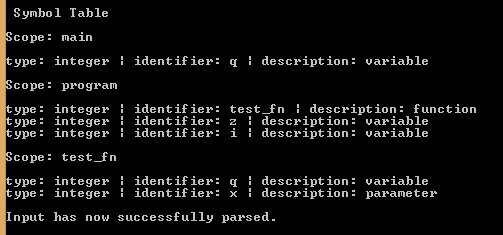
After researching the topic of symbol tables and how to implement them online,I learned that using Hash Tables and Linked Lists would of been the way to go. Firstly, I made a new file called SymbolTable.java. I used three Hash tables in my file to help identify the scope, the type ( boolean, integer etc) the description ie. if it is a variable, parameter, constant etc. After researching online, I learned that the symbol table requires an insert method.I implemented a method for this purpose called ‘Put’. For the Variables and functions etc to be inserted correctly into the table, ‘Put’ had to be invoked in several of the instances in the grammar. The ‘put’ method checks if the scope it is looking at has already been made and if it hasn’t it creates a new LinkedList for the scope and adds the declarations. They are then added to the main Hash Table with its scope and the associated identifiers. If the scope does exist it adds the new id. The id and scope are entered as one string into the description Hashset with the information string and the same procedure with the id and scope with type for the type table. I also used a ‘get’ method for retrieving information from the symbol table.



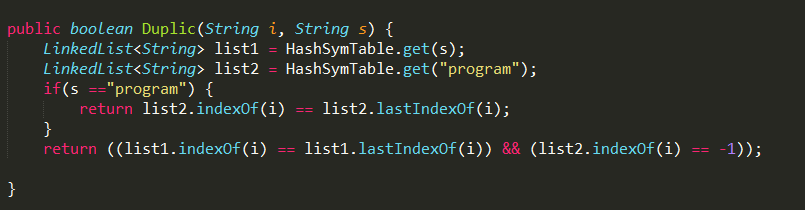
The symbol table is printed using the ‘tablePrint()’ method which is invoked from the CCAL file.

When my symbol table ran properly it showed the global scope which I labelled ‘program’, the scope for the main, and the scope for any functions in the file. See below an image of a code in the CCAL language and its corresponding symbol table.





I knew that for Semantic analysis I would be analyzing if the scopes had any duplicates. Because of this, I included a method in the symbol table file that analyzes if there are duplicate declarations, ie. if it is the main, it will check that and the program scope and if not, It will only check the program scope.



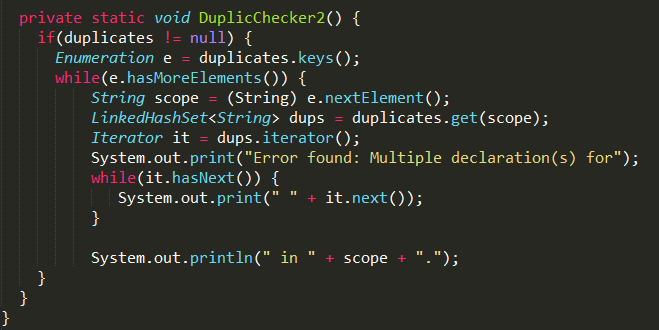
**Semantic checks**

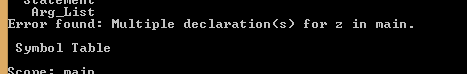
Now that the Syntax tree and the Symbol table had been properly Implemented, designing the Compilers semantic checks was next. To begin with I had to create a new file or class that implements the CCALVistitor interface which was generated by the jjtree file. My Visitor file is called ‘Analyzer.java’. This Analyzer file contained Object instances for each element of the Abstract Syntax tree. In Analyzer, a method was made for each node so they could be visited and thus semantic checks could be performed. This analyzer is initialized in my jjt file.

*Check for duplicates*

The first semantic check I included in my compiler was for duplicates of variables appearing in the same scope. I assumed that if a variable were declared in the ‘program’ scope it could not be declared again in the main or in a function. For this check I put a Hashtable in the Analyzer file called duplicates and two new methods in the Analyzer file with the node methods. The first method, ‘DupChecker1’ checks if there are duplicates in any of the scopes, this is done using the method that analyzes the Symbol Table, ‘Duplic’ . The ‘DuplicChecker1’ is invoked in the ‘ASTConstant\_Declaration’ and the ‘ASTVariable\_Declaration’ nodes in the Analyzer file. If the Symbol Table does contain duplicates they are stored in a LinkedHashSet. This is then used in the duplicates HashTable along with a string for the scope. The ‘DupChecker2’ method analyzes ‘duplicates’ and prints them out if there were any identified in the program.

Below is an image of the ‘DuplicChecker2’ method followed by one of its result error messages that is triggered when a variable ‘z’ is declared in the main and in the program scope.





**Conclusion**

Due to time constraints, time spent on other assignments and preparation for exams, I have come to the deadline with only the above work done to hand. Given the difference in weight for this assignment compared to the coming exams I have decided to submit my assignment as is. I have learned quite a few things developing the Abstract Syntax tree, Symbol table and Semantic Check for this. Although I am disappointed that this assignment is not as well done as I would like, I hope that further study of this topic for my exams may fill the gap that has been left after this assignment.

**Sources of information**

[**https://javacc.org/doc**](https://javacc.org/doc)

[**https://www.youtube.com/watch?v=8\_o9T\_aJ\_zY**](https://www.youtube.com/watch?v=8_o9T_aJ_zY)

[**https://stackoverflow.com/questions/13902239/how-to-implement-jjtree-on-grammar**](https://stackoverflow.com/questions/13902239/how-to-implement-jjtree-on-grammar)

[**https://www.youtube.com/results?search\_query=symbol+tables**](https://www.youtube.com/results?search_query=symbol+tables)