Declaration on Plagiarism

Assignment Submission Form

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

Name(s): Jaime Kirk
Programme: Compiler Construction
Module Code: CA4003
Assignment Title: Semantic Analysis and Intermediate Representation
Submission Date: 13/12/2020
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): Jaime Kirk_____ Date: __13/12/2020_____

This assignment was to add semantic analysis checks and intermediate representation generation to the lexical and syntax analyser we implemented in our first assignment.

In antlr4 the parse tree can be used as an AST so this was already generated from work done in the previous assignment.

After this I needed to create a symbol table that could handle scope. To do this I created a SymbolTableEntry class which defines the variables an entry to the symbol table has. This was based off an example the course website. These are id, type, and scope. An example of a entry would be result = [result: integer|global]. This class also had a toString() method in it so that the entries would print correctly and would be easier to use for testing. After creating a class for the entries to a symbol table, I needed to define the main symbol table class which was made up of

```
HashMap <String,LinkedList<SymbolTableEntry>> map;
Stack<String> stack;
```

The hashmap is made up of a string which is the key that will be used by get method to find existing entries to the Symbol Table. It also has a linked list of all existing entries to the symbol table. There is also a stack will store the id of entries that are added to the HashMap and scope will also be added/removed when a new scope is created/deleted. The symbol table also includes a get and put method that are used to add or retrieve an entry from the symbol table. The put method first checks if the entry also exists and if not, it will add the entry to a new entry of the hashmap. SymbolTable.java also has open and close scope methods which will add and remove scope. I also added print statements to help with testing.

Next was to implement semantic checks. I created the visitor file SemanticAsgnVisitor.java. Then the first step was to call the visitor in the assign.java file by:

```
SemanticAsgnVisitor asgnV = new SemanticAsgnVisitor();
asgnV.visit(tree);
```

The visitor extends assignBaseVisitor and all functions return Boolean values. The first thing to do was create a new symbol table for this visitor. Then I started with the visitAssignStm method which got the id, String id = ctx.ID().getText();, checked if it was in the symbol table already and if it wasn't then it had not be declared before so we print "System.out.println("Error: " + id + " not declared before use");".

Next was the constant and variable declarations. These were quite similar methods where I first declared the scope as "global" then get the id and type using

```
String id = ctx.ID().getText();
String type = ctx.type().getText();
```

Then it checks to see if the variable/constant was already declared in the scope by checking if current scope is equal to the scope of the entry in the symbol table. If it is then an error message is printed “Error: " + id + " already declared in scope”. If the constant/variable hasn’t been declared then it is added to the symbol table “symbolTable.put(id,type,scope);”.

For the nemp_paralst I stored all the ids and types of the para_lst into two separate lists using

```
List<TerminalNode> lstIds = ctx.para_lst().nemp_paralst().ID();
 List<assignParser.TypeContext> lstTypes = ctx.para_lst().nemp_paralst().type();
```

Then iterated through each of these lists using Iterator to save doing multiple while loops.

```
Iterator<TerminalNode> iterID = lstIds.iterator();
Iterator<assignParser.TypeContext> iterTypes = lstTypes.iterator();
```

Then then for each id and type add to the symbol table “symbolTable.put(String.valueOf(iterID.next()), iterTypes.next().getText(), "function_param: " + ctx.ID().getText());”

This was all the semantic checks I managed to get working. I found this assignment very difficult because of my lack of experience working with compilers and antlr and found there was little resources to help clarify how to use antlr functions and implement the semantic checks. The assignment took a long time to understand how to set up my functions and working out how to use the symbol table correctly took me longer than I would have liked, and I feel like that took away from time that I could have been trying at getting semantic checks to work.
This also meant that I was unable to make a good attempt at the IR generator as I did not have enough time to get anything working.