Name	Jordan Healy
Student Number	13379226
Programme	CASE
Module Code	CA4003
Assignment Title	Semantic Analysis and Intermediate Representation for the CCAL Language
Submission Date	Monday 12th December 2016

1 CONTENTS

2 Introduction 3
3 Abstract Syntax Tree 3
4 Symbol Table 5
5 Semantic Analysis 6
6 3-Address Code 7
7 How to run 7

2 INTRODUCTION

In this report I will discuss the following sections.

1. Abstract Syntax Tree
2. Symbol Table
3. Semantic Analysis
4. 3-address code

3 ABSTRACT SYNTAX TREE

To create the AST we decorate each production rule with a name and a number of child nodes. For example, the rule functionBody will always have 3 child nodes, declList, statementList and a return statement. Therefore we use #FunctionBody(3).

I've refactored rules that call another rule zero or more times. For example, (function())* will be put in its own rule called functionList with the decoration #FunctionList.

binary_arith_op() can perform two different actions.

If an assignment statement of a negative identifier, there should be only one child node	If it's a calculation, there should be two child nodes
…… | Assignment | | ID | | Negative | | | ID	…… | Assignment | | ID | | Subtract | | | ID | | | ID

To fix this we put <MINUS> fragment() into fragment() and decorate that node as #Negative, and then give the #Subtract decoration to the condition in binary_arith_op(). (Also do the same for <PLUS>).

Each decoration can be seen easily in the Assignment2.jjt file. The AST is generated in the Assignment2.jjt files user code by calling root.dump(""); on the parser.

Example output below;

```
Program
 DeclList
 FunctionList
  FunctionDecl
   Type(integer)
   ID(add)
   ParameterList
    Param
     ID(x)
     Type(integer)
    Param
     ID(y)
     Type(integer)
   FunctionBody
    DeclList
     Var
      ID(z)
      Type(integer)
    Return
     ID(x)
 Main
  DeclList
  StatementList
```

4 Symbol Table

The symbol table is generated in the SemanticCheckVisitor. Each symbol in the symbol table is saved as a STC object. Each object has the following components;

- Token name
- Token type
- DataType datatype
- String scope;
- LinkedHashMap<String, Object> values
- int numArgs = -1;
- boolean isRead
- boolean isCalled

DataType is an enumeration of the different types of symbols in the table, function, variable or constant.

To help populate the Symbol Table we refactor some production rules again to return Tokens. For example, we create a rule called identifier() which will return that token for every call of identifier().

In the SemanticCheckVisitor we create a data structure HashMap of HashMaps for the Symbol Table. An example of the symbol table can be seen below;

```
{ "Program":
  { "x": STC[x, integer, Variable, "Program" {x=2}, -1, true, false],
    "y": STC[y, boolean, Constant, "Program" {y=true}, -1, false, false],
    ... },
  ...
}
```

The outer map contains the key of the scope and the value is the symbols in that scope. The inner map has the key of the name of the symbol, and its corresponding STC object.

The Symbol Table is displayed in the SemanticCheckVisitor.java file after visiting every node in the AST.

Sample output is as follows;

```
**** Symbol Table ****
Scope: Main
i
 DataType: Variable
 Type: integer
 Initial Value: {i=1}
 Is written to?: true
 Is read from?: true

**********************
```

5 SEMANTIC ANALYSIS

The following checks our carried out in the SemanticCheckVisitor.

- Is every identifier declared within scope before it is used?
- Is no identifier declared more than once in the same scope?
- Is the left-hand side of an assignment a variable of the correct type?
- Are the arguments of an arithmetic operator the integer variables or integer constants?
- Are the arguments of a boolean operator boolean variables or boolean constants?
- Is there a function for every invoked identifier?
- Does every function call have the correct number of arguments?
- Is every variable both written to and read from?
- Is every function called?

Along with these checks, there are some additional checks.

- Function is already declared with the same number of parameters
- Function attempting to be declared with one or more parameters with the same id
- Argument in a function call is of incorrect type
- Variable is not declared when trying to assign it a value
- Variable has no value when attempting to do an operation

Each error displays a message explaining the problem while also pointing to the line and column it is found at.

Each error is counted and the number of errors are displayed at the end of the output.

Some errors may only occur when the previous errors are addressed. For example,

```
integer add (x:integer, y:integer) {
   var z:integer;
   z = x + y;
   return x;
}
```

X and y are said not to have been read to, but this is only the case as they do not have a value yet.

Each visit method in the SemanticCheckVisitor.java file describes how many, and what type of child nodes the current node will have.

The intermediate representation will only be generated if there are no errors.

6 3-Address Code

I have created an object called AddressCode that simply holds 4 string values, for the IR.

ThreeAddressVisitor.java has a visit method for each node and will represent the ccl code in 3-address code. Sample output below;

```
**** IR using 3-address code ****
L1
 = i 2
 return
L2
 = i 1
 funcCall test_fn i
 goto L1
*********************************
```

7 How to run

The parser can be run with the following commands;

- jjt Assignment2.jjt
- javacc Assignment2.jj
- javac *.java
- java Assignment2 inputfile