DCU School of Computing
Assignment Submission

Student Name:	Kyrylo Khaletskyy
Student Number:	15363521
Programme:	BSc. in Computer Applications
Project Title:	Assignment 2: Semantic Analysis and Intermediate Representation
Module code:	CA4003
Lecturer:	David Sinclair
Due Date:	17/12/18

Declaration

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. I understand that plagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, 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.

I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.

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. I have read and understood the referencing guidelines found at http://www.library.dcu.ie/citing&refguide08.pdf and/or recommended in the assignment guidelines.

I understand that I may be required to discuss with the module lecturer/s the contents of this submission.

Signed: Kyrylo Khaletskyy

Date: 13/12/2018

Introduction

In this report, I will outline the main features of the second assignment for CA4003 Compiler Construction and explain how I obtained my results. To begin, I created a new folder called "assignment2" and copied the previous .cal test files found in the previous assignment. Also, I copied the CALParser.jj file which I was working on during the last assignment and changed the file extension to .jjt. I changed the user code section as shown in the video and example files provided on the course website.

AST

```
//AST
System.out.println("***ABSTRACT SYSTAX TREE***");
SimpleNode root = parser.program();
root.dump("");
System.out.println("*************************");
System.out.println();
```

For this section, I used the code which I wrote for the previous assignment and adapted it to a similar fashion found on the CA4003 course website. First I added root.dump() which prints the entire AST (shown in screenshot above). Then I started with the video on JJTree. This gave me a great starting point on how to add decorations to my current grammar.

```
SimpleNode program() #PROG: {} {
    decl_list() function_list() main()

    <EOF> {return jjtThis;}
}
```

Any node which I wanted to be added to the AST was decorated with a #NAME where "name" is whatever name I decided to call the node in the AST (for example first node is named PROG as shown above). In my options I set the NODE_DEFAULT_VOID = true;, this removes the need to write #void at the end of each method which doesn't require a node to be added to the AST. Next, as shown in the video I created a new method called Identifier() and replaced the <ID> wherever necessary. I continued adding decorations to the entire grammar in a similar fashion. Within comp_op I took out expression() as I didn't want to parse the expression here, instead just the operator. I added a couple of helper methods such as comparison() which moved expression() comp_op() expression() into a separate method, which makes it easier to build the tree. And lastly, I added (2) after ORCOND and ANDCOND to ensure that they both have two children. Next, I tested my new grammar against the same .cal test files as for assignment one.

Sources:

https://www.computing.dcu.ie/~davids/teaching.shtml
https://github.com/theodore-norvell/the-teaching-machine-and-webwriter/blob/master/trunk/tm/src/tm/javaLang/parser/JavaParser.jjt

Symbol Table

```
//SYMBOL TABLE
System.out.println("******SYMBOL TABLE********");
ST.printTable();
System.out.println("*************************");
System.out.println();
```

For this section I added ST.printTable which prints the entire Symbol Table (as shown in screenshot above). This method calls the STC class. This class contains two methods, an add() method which adds items to a hashtable and a printTable() method which prints this table in the terminal. After some research (link below) I found that most symbol tables are done using a LinkedList within a hashtable. Here I used a java hashtable chaining example found at the link below and implemented it within my STC class. Each type, object and ID is entered in their respective hashtables. Later when the printTable() method is called these hashtables are iterated though and printed in the terminal.

Sources:
http://bigdatums.net/2016/07/19/how-to-create-a-hash-table-in-java-chaining-example
http://web.cs.iastate.edu/~weile/cs440.540/5.SemanticAnalysis.scope.pdf
https://www.dreamincode.net/forums/topic/206807-print-all-keys-value-pairs

```
void var_decl() #VARDECL : {Token t; String id; String type;} {
    t = <VAR> id = Identifier() <COLON> type = type()

    {
        jjtThis.value = t.image;
        ST.add(id, type, "var", scope);
    }
}
```

After I implemented the STC class I called the ST.add(); method from the STC class in order to add each entry into the hashtable as shown above. This method was called anywhere the identifier() method was called. At the beginning of the parser the scope is set to "global", but anytime the scope changes the String variable scope is set to a new string depending on what the scope currently is and added to the hashtable.

Conclusion

In conclusion, I am very disappointed with this assignment, I couldn't complete any other parts of this assignment other than AST and Symbol Table. I found this assignment very challenging and even the first two parts took a lot of time to complete, and I cannot afford to spend more time on this assignment.

Nevertheless, below is an example output I got for AST and Symbol Table, using the functions2.cal found in my test folder. Note, that is not the full output for the AST, the screenshot is shortened down for demonstration purposes.

```
***ABSTRACT SYSTAX TREE***
PROG
 FUNCTION
  TYPE
  IDENT
  PARAM
   IDENT
   TYPE
   PARAM
    IDENT
    TYPE
  VARDECL
   IDENT
   TYPE
  VARDECL
   IDENT
   TYPE
  ANDCOND
   COMPARISON
    FRETURN
    LTComp
    NUMBER
   COMPARISON
    FRETURN
    GTEQComp
    NUMBER
  IDENT
  TRUEOP
  ASSIGN
  IDENT
  IDENT
  MINUSOP
  ASSIGN
  ANDCOND
   COMPARISON
    FRETURN
    LTComp
    NUMBER
   COMPARISON
    FRETURN
    GTEQComp
    NUMBER
  IDENT
  TRUEOP
  ASSIGN
```

```
******SYMBOL TABLE********
main:
     const      five    integer
     var        result  integer
     var        arg_2   integer
     var        arg_1   integer
multiply:
     var        minus_sign      boolean
     var        result  integer
     param      x       integer
     param      y       integer
global:
     func       multiply        integer
*************************
```