Compilers Assignment 2 CA4003

Shannon Mulgrew 16304263

15/12/19

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): _____Shannon Mulgrew_____ Date: ___15/12/19_____

Introduction

This assignment is a continuation of assignment 1 where we had to implement a lexical and syntax analyser for the ccal language
Firstly I started by changing the original assign1.jj file to assign2.jjt. I copied the test files from the first assignment also. The user code was taken from the video and notes provided.

Before beginning assignment 2 I had to improve my code from assignment 1 to make sure I could test it properly.
After making some changes i began by adding in user code to print the AST and Symbol Table.

```
try{
SimpleNode root = parser.program();
System.out.println(".....");
System.out.println("Abstract Syntax Tree:");
System.out.println(".....");

root.dump(" ");

System.out.println();
System.out.println(".....");
System.out.println("Symbol Table:");
System.out.println(".....");
ST.print();
```

The user code was taken from the course notes/video for taking in the test files and print the AST and Symbol table. The root.dump works with the simpleNode.java and the ST.print() works with the STC.java file.
No changes were needed in the token declaration section.
The grammar rules are where most of the changes were made

Abstract Syntax Tree

To view the tree you can use root.dump(" ") which print the entire AST.
Any node that was to be added to the AST got a name in the form of "#NAME". I have added some decorations to the code to help build an abstract syntax tree.

```
Abstract Syntax Tree:
 Program
  decl_list
  function_list
  main
   decl_list
   statement_block

Symbol Table:
```

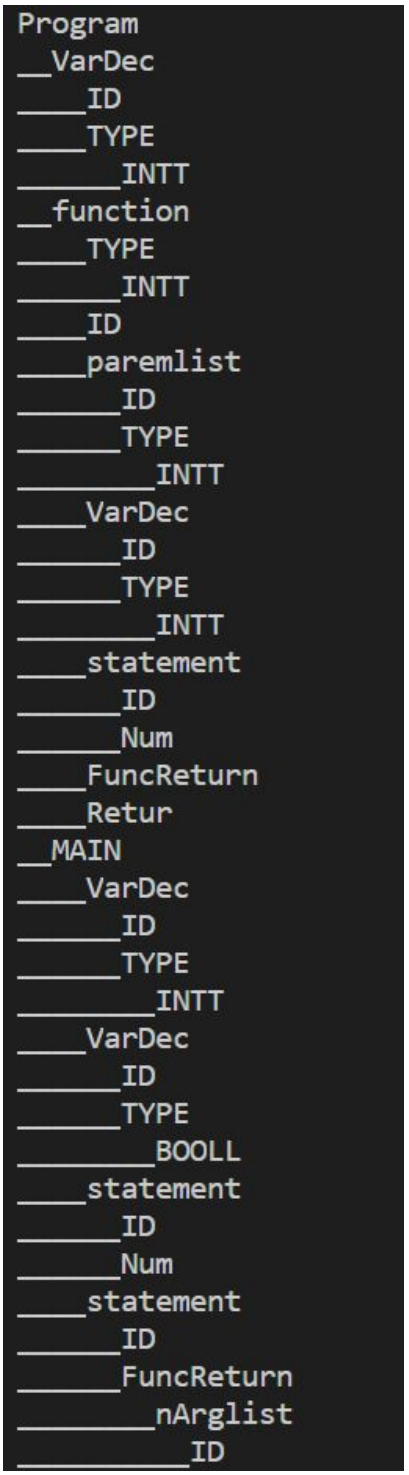
This is what was outputted after I added a node in the first grammar rule using a simple test file from assignment 1.

```
// chnaged from void to SimpleNode
SimpleNode program() #Program : {} {
    decl_list() function_list() main() {return jjtThis;}
}
```

Once I had added decorations to any grammar that needed it my tree looked like this when tested.
Once I tested the AST with a longer test code I received this AST.

Test code :

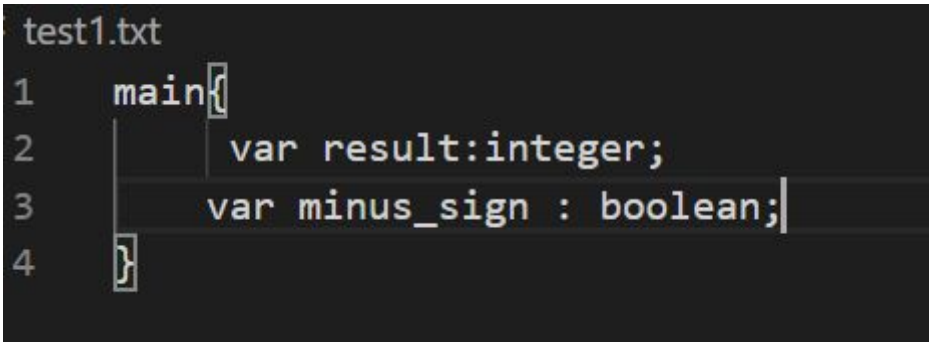
```
var i:integer;

var i:integer;

integer test_fn (i:integer){
    var i:integer;
    i=2;
    return(x);
}

main{
    var i:integer;
    var j:boolean;
    i=1;
    i=test_fn(i);
}
```



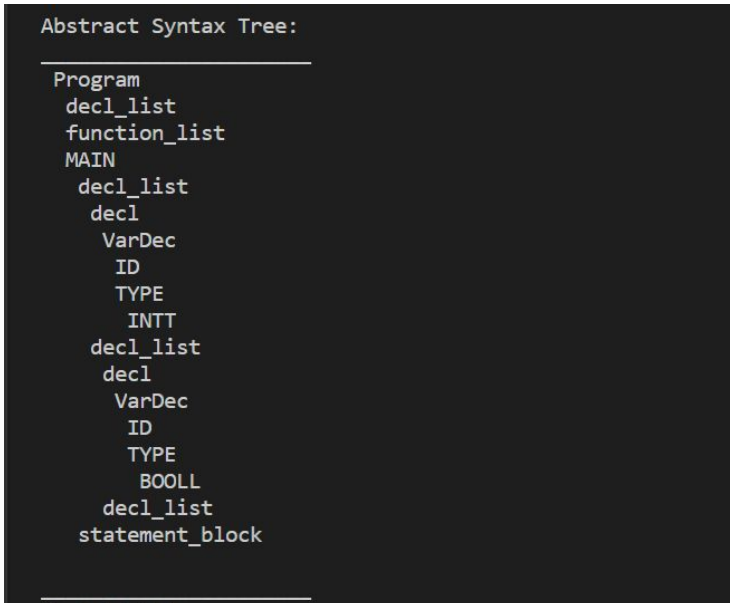
Symbol Table

When it came to the symbol table I had to go in and edit my functions so it would return the type and value of the variable and store it in a hash table to be printed when running a test file.
The jjtThis is a pointer to an AST node.
It took a var of token and returned it to the STC.java file.
The STC.java file held a hashmap that all the variables and their description were stored in. The grammar rules sent back the id var and scope of a variable as it ran through the grammar files.

When I ran the test code on this file.



I received this AST and Symbol Table. This part of the assignment works with the STC.java file onto hold all values for variables and name and types.



When I ran this test code against my parser it returned this symbol table as well as the AST below.

```
var i:integer;

integer test_fn (x:integer){
    var i:integer;
    i=2;
    return(x);
}

main{
    var i:integer;
    i=1;
    i=test_fn(i);
}
```

```
Symbol Table:
_____________________________
|         ID |       Type |      Scope |  Description|
-------------------------------------------------------
|          i |    integer |       main |     variable|
|          i |    integer |     global |     variable|
|          i |    integer |    test_fn |     variable|
|          x |    integer |    test_fn |        param|
|    test_fn |    integer |    test_fn |         func|
```

Semantic Analyser

For the semantic analyser I would have followed the video provided and created visitors to to print the AST and a type checker to check addition/subtraction etc. The visitors were to check the AST tree to make sure it was printing correctly

and the type Checker visitor would have been to check the addition/subtraction etc of the AST.

Intermediate Representation

The Three address code would have acted similar to the visitor talked about in the course video. It would have visited each node recursively. This IR is needed to optimise the code and make it more efficient for each IR there are certain rules to follow for the three address code you had to have at most one operator on the right hand side of an instruction as said in the notes.

Conclusion

In conclusion, I found this to be a very difficult assignment and improvements and changes could be made to make it work better and more efficiently. Even though I struggled to wrap my head around how everything works I found it to be very helpful in understanding how compilers work and the theory behind it.

Links

https://web.cs.wpi.edu/~cs544/PLTprojectast.html

https://www.youtube.com/watch?v=9V4GzEomc5w
https://www.engr.mun.ca/~theo/JavaCC-Tutorial/javacc-tutorial.pdf
http://www.cs.um.edu.mt/~sspi3/CSA2201_Lecture-2012.pdf?fbclid=IwAR0U4IitOveuKhIRoQU9TqDV0pGkmcLfLHNqtyeLzRuklaIUzhPcRKarLBo
https://www.computing.dcu.ie/~davids/courses/CA4003/CA4003_JJTree_2p.pdf?fbclid=IwAR29mCSaMYcfSt7tT3ehw_fdpmjP6WLBAGJzMymjgQBEPgJFwk43lkQLfU8
https://stackoverflow.com/questions/53345281/how-do-i-build-an-abstract-syntax-tree-with-jjtree?fbclid=IwAR0OzTyzgFBm7uCw-muKZALXLwZZwyL31IfVwyqnpk99Tceyb41NdD5uF1c