

CA4003 Compiler Construction ASSIGNMENT2

|  |  |  |
| --- | --- | --- |
| **Student Name** | **Student ID** | **Course Code** |
| Nathan Ndombasi | 13517227 | CASE4 |

Declaration:

*I declare that this material, which I 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 is a grave and serious offence in the university and accept the penalties that would be imposed should I (we) engage in plagiarism, collusion, or copying. I (we) have read and understood the Assignment Regulations set out in the module documentation. 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 have read and understood the referencing guidelines found at http://www.library.dcu.ie/citing&refguide08.pdf and/or recommended in the assignment guidelines. “*

## Aim

The aim of this assignment was to implement to add semantic analysis checks and intermediate representation generation to the lexical and syntax analyser you have implement in Assignment 1. The generated intermediate code should be a 3-address code and stored in a file with the ".ir" extension.

### Introduction

For this section of my assignment, I wrote the Semantic Analysis (EvalVisitor.java) and the Intermediate Representation (IRVisitor.java) phase of the compiler. Assignment2 is a derivation of Assignment 1 where I had to generate AST (Abstract Syntax Tree) which is made up of Symbol Table, Semantic Checks, and Intermediate Code.

The function of the Symbol Table (SymbolTable.java) is to store each token in a Hash Table which is passed to the IRVisitor.

Unfortunately, I was unsuccessful with generating an intermediate code into an output file (outputFile) as my ANTLR4 stopped initialising my main “cal”. I tried to re-download antlr4 on my laptop and run my cal files again, but it did not work. However, all my java files compiled successfully and the last error I received was the Tree was returning null.

The EvalVisitor.java is used to check if there was a semantic error and if it found one it would print out one according to the error.

### Cal.g4 Grammar File

I worked with the Grammar file that I wrote in Assignment1 but for this assignment we had to extend it by writing an alias for each method which is then used by the EvalVisitor to traverse through.

### EvalVisitor.java(Semantic Check)

The purpose of the EvalVisitor.java is to provide a set of rules based on syntax structures. It calls the Symbol Table at the beginning of the program and I push the types into the Symbol Table. This is where scopes are first introduced. The scopes are then permanently deleted when the tree has been fully visited.

There were several semantic checks that I had to make sure was done before overriding my functions. I followed the example of David Sinclair’s walkthrough and the previous labs held by him which were very informative. This still proved to be quite difficult, so it was a lot of trial and error to make sure it compiled successfully.

These are the semantic checks that have been placed.

1. Assigning a value to variable (checks if it is a variable & if the type matches)
2. When calling a function (has this function been declared? If so is it the same type)
3. Logical Operators of are of type Boolean.
4. Binary arithmetic are of type int.
5. Declaring a constant is the value assigned the same type.
6. Declaring a function is the returned the same type,

If there was a semantic error, then my program would System.out.println a message indicating where that error was.

If there was no semantic error, then it would print out indicating it was a success.

## Cal.java

As Cal.java was extended from the first assignment. This is where I introduced the EvalVisitor EvalV so it can visit the tree. If it traversed the tree successfully it would print out that it was a success. This is also the same case for the IRVisitor.

## IRVisitor.java

For the implementation of my IRVisitor, I followed similar steps to the EValVisitor. I focused on the methods that were influenced by the command line and returned them accordingly and pushed them into the stack.

***fin.***