# CA4003 - Assignment 1

## Plagurism Statement

**Name:** Alan Devine

**Student ID:** 17412402

**Programme:** CASE4

**Module Code:** CA4003

**Assignment Title:** A Lexical and Syntax Analyser

**Submission Date:** 15/11/2020

**Module Coordinator:** David Sinclair

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 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 have read and understood the Assignment Regulations. I 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 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.dcu.ie/info/regulations/plagiarism.shtml , https://www4.dcu.ie/students/az/plagiarism and/or recommended in the assignment guidelines.

**Name:** Alan Devine
**Date:** 15/11/2020

# CAL.g4

This grammar file follows the Language Specification quite faithfully. This file is laid out in such a way that the more abstract components of the language are defined first. The starting rule, or, the highest level non-terminal is `prog` and it is defined as so.

```
prog
    :   decl_list function_list main
    ;
```

The only additional rule/ non-terminal that I needed to add was `empty_statement`. This is the equivalent of ε as found in the language definition, and is defined as the following.

```
empty_statment
    :
    ;
```

# cal.java

cal.java is the main process and is used as such.

```
foo@bar:~$ java cal file.cal
file.cal parsed successfully
```

This program works by taking in some program written in the cal programming language, we then construct a *Lexer*, a *Token Stream* and a *Parser*. This is accomplished using the following.

```
CALLexer lexer = new CALLexer(CharStreams.fromStream(is));
CommonTokenStream tokens = new CommonTokenStream(lexer);
CALParser parser = new CALParser(tokens);
```

There are two notable additions to instantiating the parser. By default, when `lexer` and `parser` are instantiated there is a certain number of predefined error listeners present as a part these Object. These error listeners print to `System.err` and while this wouldn't typically be an issue, because the output of this program is to `System.out`, these predefined error listeners need to be removed. After removing all the default error listeners, a custom built error listener, `CALSyntaxListener`, is added.

```
lexer.removeErrorListeners();
lexer.addErrorListener(new CALSyntaxListener());
...
parser.removeErrorListeners();
parser.addErrorListener(new CALSyntaxListener());
```

With this new *Listener* in place, we can wrap `parser.prog()` in a try/ catch block.

```
try {
    parser.prog();
    System.out.println(inputFile + " parsed successfully");
} catch (Exception e) {
    System.out.println(inputFile + " has not parsed");
}
```

## CALSyntaxListener.java

This class extends the BaseErrorListener class in the ANTLR library. The purpose of this listener is throw a `ParseCancellationException` when the parser encounters errors in the parsing. For the sake of transparency, while implementing this listener, I referred to "The Definitive ANTLR 4 Referenced", specifically page 154.

The only action this listener takes is to throw the following exception.

```
throw new ParseCancellationException("line " + line + ":" + charPositionInLine +
" " + msg);
```