CA4003 Assignment 1

Joao Pereira / 19354106

Declaration on Plagiarism

Assignment Submission Form

This form must be filled in and completed by the student(s) submitting an assignment

Name(s): Joao Maria Baeta Pereira
Programme: Computer Applications & Software Engineering
Module Code: CA4003
Assignment Title: A Lexical and Syntax Analyser
Submission Date: 1st of November 2022
Module Coordinator: David Sinclair

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): __Joao_Maria_Baeta_Pereira_____ Date: ____31_/10_/22_____

Implementation

I created the required files to create a syntax and lexical analyser for the CAL language. These include a file.g4, file.java and a few sample test files.

file.g4

This is an Antlr Grammar file which is responsible for all the grammar, logic and necessary operators needed to provide an output for the CAL language. This file holds grammar such as the fragments, tokens and lexer and parser rules. The grammar was written based on the Construction Assignment Language Definition. Fragments were first generated for each letter used within CAL. Following, all the required reserved words, operators and syntax tokens are implemented. And lastly, parser rules are developed to provide the logic to Construction Assignment Language.

file.java

This java file holds the responsibility to print an output indicating whether the parser has failed or passed. This file is based on the provided example from the lecture materials with a few added tweaks to fit the required assignment needs. The tweaks include adding a try and catch method along with an error_handling file to be able to catch the success and failure as well as an error listener to handle the errors. This was created with the aid of a stack overflow page.

error_handling

An error handling java file was created as an aid to provide error handling to the main java compiler file. This file adds the ability for the compiler to be able to catch an error if there is an error. Without this error handling file, the java file would simply output the errors but it would also print success each time as it would have nothing to catch onto.

Execution

The file was ran through the command “antlr4 file.g4; javac *.java; java file ___.cal”, where the underscores would be the name of a CAL test file. To begin, the Antlr file was compiled to ensure there were no errors occurring. Once the Antlr command was executed, it then created a number of lexer and parser files within the directory. Following this, those generated java files along with the java parser were also compiled to ensure no errors were present and if so, they were fixed. Finally, I ran the mentioned command which would execute the first two steps and then test the analyser on a sample CAL file. The command either outputs “____ Parsed Successfully” if the parser has no errors or “____ has not parsed” if the Antlr file has issues with dealing with CAL.

Resolved Issues

Some of the issues I ran into whilst developing this project include:

Comments: This token caused issues due to the rules it possessed; it was soon fixed by trial and error by compiling the Antlr code, running the java files and adding regex until no more errors were outputted.
Error Handling: Error handling in general was quite tough to implement however through the use of online resources I was able to implement and understand the overall concept as a whole.
Parser and lexical rule: A Lot of errors occurred when running the test files however most of them were due to either typos or forgotten operators.
Mutually Left Recursive: This was an issue caused by the fragment parser rule. I managed to fix this issue with the use of the module materials as well as online resources. This was accomplished by replacing expression from the fragment2 parser rule to ID LBR arg_list RBR which is one of the rules from the expression parser rule.

References

Sources outside of lecture materials utilised in aid of completion of project:
https://stackoverflow.com/questions/18132078/handling-errors-in-antlr4
https://stackoverflow.com/questions/41017948/antlr4-the-following-sets-of-rules-are-mutually-left-recursive
https://stackoverflow.com/questions/7070763/parse-comment-line
https://stackoverflow.com/questions/12898052/antlr-how-to-skip-multiline-comments