CA4003 Compiler Construction

Assignment #1 Lexical and Syntax Analyser

Olan Buckeridge

15461022

Declaration on Plagiarism

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 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/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.dcu.ie/info/regulations/plagiarism.shtml , https://www4.dcu.ie/students/az/plagiarism and/or recommended in the assignment guidelines.

Name: Olan Buckeridge
Date: 10/11/2018

Overview

The aim of this assignment is to implement a lexical and syntax analyser using JavaCC for a simple language called CAL.

The language is not case sensitive. A nonterminal, X, is represented by enclosing it in angle brackets, e.g. hXi. A terminal is represented without angle brackets. A bold typeface is used to represent terminal symbols in the language and reserved words, whereas a non-bold typeface is used for symbols that are used to group terminals and nonterminals together. Source code should be kept in files with the .cal extension, e.g. hello world.cal .

Syntax

The reserved words in the language are **variable, constant, return, integer, boolean, void, main, if, else, true, false, while, begin, end, is** and **skip**. The following are tokens in the language:
, ; : := ( ) + - ~ | & = != < <= > >=
Integers are represented by a string of one or more digits (‘0’-‘9’) and may start with a minus sign (‘-’), e.g. 123, -456. Unless it is the number ‘0’, numbers may not start with leading ‘0’s. For example, 0012 is illegal. Identifiers are represented by a string of letters, digits or underscore character (‘_’) beginning with a letter. Identifiers cannot be reserved words. Comments can appear between any two tokens. There are two forms of comment: one is delimited by /* and */ and can be nested; the other begins with // and is delimited by the end of line and this type of comments may not be nested.

Semantics

Declaration made outside a function (including main) are global in scope. Declarations inside a function are local in scope to that function. Function arguments are passed-by-value. Variables or constants cannot be declared using the void type. The skip statement does nothing.

Description

For the lexical analysis I began with defining all of the valid tokens that the compiler would accept. I split these tokens into three different sets. Firstly I input the reserved keywords. The next set I focused on were the operators and punctuation of CAL. Finally I added the acceptable identifiers of the language. Once that was done I focused on all of the elements I wanted the compiler to skip. I needed to skip comments (single-line and multi-line) followed by spaces, tabs, carriage returns and new lines.

For the syntax analyser, there was a few conflict choices and left-recursion errors for CAL. Most of these choices and errors were taken care of by using factorisation. This was solved by taking common parts of conflicting alternatives and transferring them to the front. Regular expression syntax was also used to allow for iteration as opposed to recursion. When I ran the javacc file it was unable to parse test files due to lookaheads. I found this part of the assignment the most difficult to overcome. I separated the functions and started trying to look at each problem individually. I was able to stop where I was receiving my warnings my manually rearranging the productions and duplicating them then || in the expression().

In conclusion I found the assignment quite challenging and required a lot of thinking to deal with the warnings and lookaheads. I am now more comfortable with dealing with the difficulties of problems with lexical and syntax analysers.

Testing

For testing I created test files from
https://www.computing.dcu.ie/~davids/courses/CA4003/cal.pdf