Declaration on Plagiarism

Assignment Submission Form

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

Name(s): Alex Thornberry
Programme:Computer Applications Software Engineering
Module Code: CA4003
Assignment Title: A Lexical and Syntax Analyser for the CCAL Language
Submission Date: 03/11/2019
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): Alex Thornberry Date: 03/11/2019

AlParse Description

I began this assignment by utilizing the started code within this modules notes. This meant for Section 1 the only option I began with was JAVA_UNICODE_ESCAPE = true;. I later added IGNORE_CASE = true; as the language is case insensitive.

I again utilized the modules notes to write my section 2 code. I removed the block of code that printed the tokens and added in the code that began the program i.e. parser.Prog()

In section 3 I had to define the tokens, this included skips, comments, reserved words, operators and numbers and identifiers. I completed this using the given syntax in the ccal language definition.
For example;
One of the immediate challenges was as follows;
“Integers are represented by a string of one or more digits (‘0’-‘9’) that do not start with the digit ‘0’, but may start with a minus sign (‘-’), e.g. 123, -456”

```
< INT : "0" /*CAN BE 0*/
| ("-")? ["1"-"9"] (<DIGIT>)* > /*CANT START WITH 0*/
```

This was easily implemented using the OR operand and the optional operand “?”.

I then completed the rest of the code as per the guidelines set out. Once I had finally transcribed all of the grammar I compiled the jj file.

Left recursion

Upon running the javacc command on my jj file which was then called assignment1.jj, I encountered the following errors;

```
C:\Users\Alex Thornberry\College\Compilers - ca4003\assignment1>javacc assignment1.jj
Java Compiler Compiler Version 6.0_1 (Parser Generator)
(type "javacc" with no arguments for help)
Reading from file assignment1.jj . . .
Error: Line 226, Column 1: Left recursion detected: "Expression... --> Fragment... --> Expression..."
Error: Line 250, Column 1: Left recursion detected: "Condition... --> Condition..."
Detected 2 errors and 0 warnings.
```

Direct Left Recursion:

The first error I set out to remove was the direct left recursion in Condition. As seen below, Condition has the chance to call itself every single time.

```
250  void Condition() : {}
251  {
252    <TILDA> Condition()
253  | <LBRAC> Condition() <RBRAC>
254  | Expression() CompOp() Expression()
255  | Condition() (<LOG_OR> | <LOG_AND>) Condition()
256  }
257
```

The notes given to us in our notes included a very helpful formula and upon following the formula

```
250  void Cond_Prime() : {}
251  {
252      (<LOG_OR> | <LOG_AND>) Condition() Cond_Prime()
253      |   {}
254  }
255
256  void Condition() : {}
257  {
258    <TILDA> Condition() Cond_Prime()
259  | <LBRAC> Condition() <RBRAC> Cond_Prime()
260  | Expression() CompOp() Expression() Cond_Prime()
261  }
```

```
C:\Users\Alex Thornberry\College\Compilers - ca4003\assignment1>javacc assignment1.jj
Java Compiler Compiler Version 6.0_1 (Parser Generator)
(type "javacc" with no arguments for help)
Reading from file assignment1.jj . . .
Error: Line 226, Column 1: Left recursion detected: "Expression... --> Fragment... --> Expression..."
Detected 1 errors and 0 warnings.
```

Indirect Left Recursion

The next left recursion error was indirect recursion, as seen below Expression can call Fragment which can call expression and the this can become infinite.

```
void Expression() : {}
{
  Fragment() Binary_Arith_Op() Fragment()
| <LBRAC> Expression() <RBRAC>
| <ID> <LBRAC> Arg_List() <RBRAC>
| Fragment()
}

void Fragment() : {}
{
  <ID>
| <MINUS_SIGN> <ID>
| <INT>
| <TRUE>
| <FALSE>
| Expression()
}
```

To remove this, I followed the formula given in our notes for eliminating indirect left recursion. This led to me creating two prime methods, one for Expression() and one for Fragment().

This brought us to move from indirect left recursion to direct left recursion. We could then reapply the algorithm for eliminating left recursion and that left us with the below;

```
void Expression() : {}
{
  <LBRAC> Expression() <RBRAC> exp_prime()
| <ID> <LBRAC> Arg_List() <RBRAC> exp_prime()
| Fragment() exp_prime()
}

void exp_prime() : {}
{
  Binary_Arith_Op() Fragment() exp_prime()
| {}
}

void Binary_Arith_Op() : {}
{
  <PLUS_SIGN>
| <MINUS_SIGN>
}

void Fragment() : {}
{
    FragmentPR()
}

void FragmentPR() : {}
{
  <ID>
| <MINUS_SIGN> <ID>
| <INT>
| <TRUE>
| <FALSE>
}
```

I then cleaned up the FragmentPR() into Fragment as there was a redundant method.

```
void Fragment() : {}
{
    FragmentPR()
}

void FragmentPR() : {}
{
  <ID>
| <MINUS_SIGN> <ID>
| <INT>
| <TRUE>
| <FALSE>
}
```

Choice Conflicts

```
C:\Users\Alex Thornberry\College\Compilers - ca4003\assignment1>javacc assignment1.jj
Java Compiler Compiler Version 6.0_1 (Parser Generator)
(type "javacc" with no arguments for help)
Reading from file assignment1.jj . . .
Warning: Choice conflict involving two expansions at
         line 198, column 3 and line 199, column 3 respectively.
         A common prefix is: <ID> ":"
         Consider using a lookahead of 3 or more for earlier expansion.
Warning: Choice conflict involving two expansions at
         line 218, column 3 and line 219, column 3 respectively.
         A common prefix is: <ID>
         Consider using a lookahead of 2 for earlier expansion.
Warning: Choice conflict involving two expansions at
         line 229, column 3 and line 230, column 3 respectively.
         A common prefix is: <ID>
         Consider using a lookahead of 2 for earlier expansion.
Warning: Choice conflict involving two expansions at
         line 268, column 3 and line 269, column 3 respectively.
         A common prefix is: "(" "("
         Consider using a lookahead of 3 or more for earlier expansion.
Warning: Choice conflict involving two expansions at
         line 290, column 3 and line 291, column 3 respectively.
         A common prefix is: <ID>
         Consider using a lookahead of 2 for earlier expansion.
File "TokenMgrError.java" is being rebuilt.
File "ParseException.java" is being rebuilt.
File "Token.java" is being rebuilt.
File "JavaCharStream.java" is being rebuilt.
Parser generated with 0 errors and 5 warnings.
```

There were no longer any errors within the code, this allowed me to view the many warnings that were present, these conflicts were caused by a method having two paths which begin

the same, thus causing the program to not know which path to take, of course these can be solved by lookaheads but we would prefer to eliminate the conflicts altogether.

The easiest conflicts to solve were as follows:

```
void Nemp_Parameter_List() : {}
{
  <ID> <COLON> Type()
| <ID> <COLON> Type() (<COMMA> Nemp_Parameter_List())?
}
```

```
void Nemp_Arg_List() : {}
{
  <ID>
| <ID> <COMMA> Nemp_Arg_List()
}
```

As you can see they are very similar both second lines being simply an extension on the first line.
To fix this we must simply merge the two lines into one line and allowing the extension to be optional.

```
void Nemp_Parameter_List() : {}
{
<ID> <COLON> Type() (<COMMA> Nemp_Parameter_List())?
}
```

```
void Nemp_Arg_List() : {}
{
  <ID> (<COMMA> Nemp_Arg_List())?
}
```

The following choice conflict was similar but I was not able to simply merge the lines make them optional statements as there was then a possibility that neither were selected, which was outside of the scope.

```
void Statement() : {}
{
  <ID> (<EQUA> Expression() <SEMICO>
| <ID> <LBRAC> Arg_List() <RBRAC> <SEMICO>
```

I had to create another method to choose between them by factoring out the <ID> which was common between both lines. Factoring that out and then calling a new method removed the choice conflict.

```
void Statement() : {}
{
  <ID> Statement_prime()
| <LCB> Statement_Block() <RCB>
```

```
void Statement_prime() : {}
{
  <EQUA> Expression() <SEMICO>
| <LBRAC> Arg_List() <RBRAC> <SEMICO>
}
```