I declare that this material, which I now submit for assessment, is entirely my own work unless otherwise stated 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 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(s): _Shannon Mulgrew_____ Date: __3/11/19_____

INTRO

In this report I will outline how I went about Assignemnt 1 which was to create a lexical and syntax analyser for the CCAL language. All the code is within the assign1.jj file.For this assignment I worked off the CCal pdf provided and the course notes. The assign1.jj file contains 4 sections.

1) *Options*
2) *User Code*
3) *Token Definitions*
4) *Grammar Rules*

---

1)OPTIONS

For the CCAL language, as it's not case sensitive I have it IGNORE_CASE=true;

```
options{
  IGNORE_CASE = true;
}
```

---

2)USER CODE

The majority of this code came from the JAVACC course notes.
To run the parser needs a declaration with its parser name.

```
PARSER_BEGIN(Assign1)
```

The user code initializes the parser if the input file is passed correctly, if not it exits with instructions on how to use the parser.
There is a try{} catch{} statement to attempt to tokenize the program. If it's successful "Parsing successful" will print, otherwise and error message will be shown.

To end the Parser you have to have `PARSER_END(Assign1)` at the end of your user code.

---

3)TOKENS

When I began this assignment I started with the token definition section first. From looking at the notes and reading through the ccal pdf I defined my tokens below along with Skip options:

As stated in the docs there were 'reserved words' for ccal. Symbols, operators and punctuation is also defined below.

Skip- Empty strings, tabs, newlines, comments etc.

Token Definitions -

/*Reserved words*/

/*Symbols and Operators*/

/*Regular Expressions*/

```
/* RESERVED WORDS */
TOKEN : {
    <VAR: "var">
   |<CONST:"constant">
   |<RETURN:"return">
   |<INTEGER:"integer">
   |<BOOLEAN:"boolean">
   |<VOID:"void">
   |<MAIN:"main">
   |<IF:"if">
   |<ELSE:"else">
   |<TRUE:"true">
   |<FALSE:"false">
   |<WHILE:"while">
   |<SKIPP:"skip">
```

```
/*Punctuation and operators*/

   |<COMMA:",">
   |<SEMICOLON:";">
   |<COLON:":">
   |<ASSIGN:"=">
   |<LBRACE:"{">
   |<RBRACE:"}">
   |<LBRACK:"(">
   |<RBRACK:")">
   |<PLUS:"+">
   |<MINUS:"-">
   |<NEGATE:"~">
   |<OR:"||">
   |<AND:"&&">
   |<EQUAL:"==">
   |<NOT_EQUAL:"!=">
   |<LESSTHEN:"<">
   |<LESSTHEN_EQUAL:"<=">
   |<GREATERTHEN:">">
   |<GREATERTHEN_EQUAL:">=">
```

Next I had to implement the Digit, char,num and Identifiers. The base of the code comes from the notes but ccal does not allow numbers beginning with 0 unless its 0 on its own. This is why for 'NUM' its starts '1' rather than '0' as we do not want eg. 0123, instead we want 123.

```
/*integers and string */
  |< #DIGIT: ["0"-"9"] >
  | < #CHAR: ["a"-"z", "A"-"Z"] >
  | < NUM: ((<MINUS>)? ["1"-"9"] (<DIGIT>)*) | ((<MINUS>)? ["0"]) >
  | < ID: (<CHAR>) ((<DIGIT>) | (<CHAR>) | "_")* >
```

When I compile my code at this stage I got this error. This error was to do with my token definitions.

```
org.javacc.parser.ParseException: Encountered " "SKIP" "SKIP "" at line 72, column 7.
Was expecting one of:
```

This was clashing with the SKIP section in the code and the fix was to rename it from "SKIP" TO "SKIPP" this is because 'SKIP' is a reserved word and I could not use it in my token definitions.

After all my token were defined I compiled my code again.

```
C:\Users\shann\Desktop\Ca4003\ca4003\shan>javacc assign1.jj
Java Compiler Compiler Version 5.0 (Parser Generator)
(type "javacc" with no arguments for help)
Reading from file assign1.jj . . .
File "TokenMgrError.java" is being rebuilt.
File "ParseException.java" is being rebuilt.
File "Token.java" is being rebuilt.
File "SimpleCharStream.java" does not exist.  Will create one.
Parser generated successfully.
```

Then I compiled all the other java files that had been created and once they compiled successfully i started on the grammar.

Javac *.java

---

4)GRAMMAR RULES

The production rules were written following the ccal.pdf file. Between the pdf and the course notes I coded all the rules.

After fixing a few syntax errors I compiled my code and got this error.

Left Recursion detected..

```
Java Compiler Compiler Version 5.0 (Parser Generator)
(type "javacc" with no arguments for help)
Reading from file assign1.jj . . .
Error: Line 169, Column 1: Left recursion detected: "expression... --> fragment... -
-> expression..."
Error: Line 187, Column 1: Left recursion detected: "condition... --> condition..."
Detected 2 errors and 0 warnings.
```

To solve this problem I went back into the notes.

In order to get rid of left recursion i need to change how my grammar is defined. I started with condition.

I found that splitting condition() into condition_1() and condition_2() solved my issue. In condition i call both condtion_1() and condition_2()

void condition():{}{

condition_1()condition_2()

To remove left-recursion, we can transform the grammar.

Consider the grammar fragment:

$$A ::= A\alpha \mid \beta$$

where α and β do not start with A.
We can rewrite this as:

$$A ::= \beta A'$$
$$A' ::= \alpha A' \mid \epsilon$$

where A' is a new non-terminal

This fragment contains no left-recursion.

```
}
void condition_1():{}{
  <NEGATE>condition()
  |LOOKAHEAD(3)<LBRACK>condition()<RBRACK>
  |(expression() comp_op()expression())
}
void condition_2():{}{
  <AND>condition()
  |<OR>condition()
  |{}
}
```

I found the left recursion for the expression function to be harder.
I took expression and call fragment in it. Within fragment instead of calling expression again and causing left recursion it calls fragment prime or (fragment_1) as I have named it.This eliminates left recursion.

After getting rid of the recursion choice conflict warnings were displayed. To get rid of these I followed the suggestions in the terminal to add LOOKAHEADS to see the following input strings.

---

5)TESTING

When I began testing i kept getting an error that didn't make sense to me.

```
Encountered errors during parsing.
Encountered " "}" "} "" at line 3, column 1.
```

I checked my tokens and I had “}” initialised and i couldn’t figure why but as I worked my way through the steps of the code I realised I had an error instead of having

void main():{}{

<MAIN><LBRACE>decl_list() statement_block() <RBRACE>

}

I had..

void main():{}{

<MAIN><LBRACE>decl_list() statement_block() <RBRACK>

}

One letter caused the error once I changed it my test passed.