DCU School of Computing

Assignment Submission

Student Name(s):	**Dovydas Baranauskas**
Student Number(s):	**15336731**
Programme:	**BSc in Computer Applications**
Project Title:	**CAL LANGUAGE PARSER**
Module code:	**CA4003**
Lecturer:	**DAVID SINCLAIR**
Project Due Date:	**12/11/18**

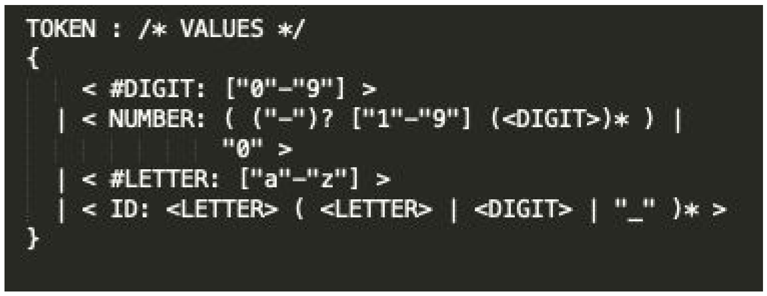
Declaration
I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying is a grave and serious offence 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 set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, 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.
I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.
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.library.dcu.ie/citing&refguide08.pdf and/or recommended in the assignment guidelines.
I understand that I may be required to discuss with the module lecturer/s the contents of this submission.
I/me/my incorporates we/us/our in the case of group work, which is signed by all of us.

Signed: Dovydas Baranausakas

CAL PARSER DESCRIPTION

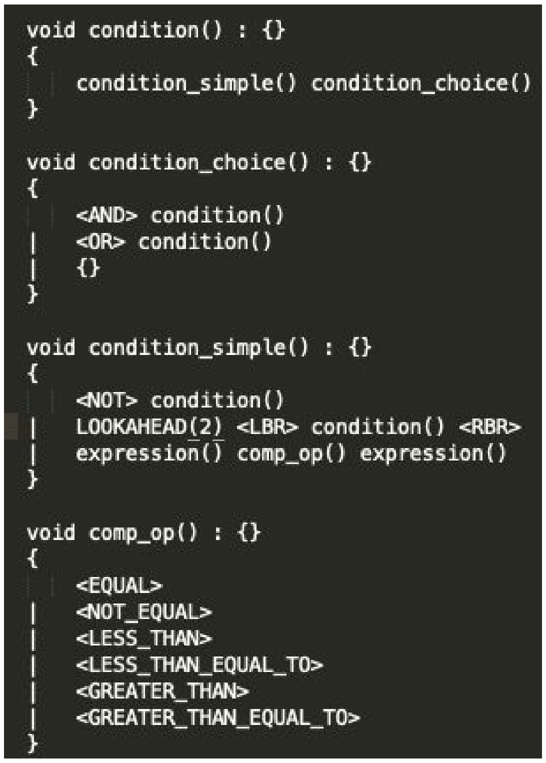
For the start of the parser I reused the code from the notes as my starting point and template. I changed the name of the parser to "CALParser" and made sure the parser.program() function was the same as my starting function, then I started working on the token analyser. I made sure to set ignore case in the options as the CAL language was not case sensitive. I used the default skip method for skipping newlines,spaces, and tabs that was in the notes. It would count up the amount of times the toke "*/" showed up and make sure the amount of "*\" matched. I just needed to add a rule to recognise single line comments as the SLPtokensier already recognises multi line comments. So the parser skips tokens after a "//" as well as after multi line comments denoted by "/* */".

I then went on to describe the keywords as described by the CAL language in the notes. After this I changed the values recognised by the language. Numbers could be negative or positive, followed by any amount of numbers from 1-9, or it could be 0. ID's must start with a letter and can be followed by a letter, digit, or underscore. This is what the values looked like.



```
TOKEN : /* VALUES */
{
    < #DIGIT: ["0"-"9"] >
  | < NUMBER: ( ("-")? ["1"-"9"] (<DIGIT>)* ) |
              "0" >
  | < #LETTER: ["a"-"z"] >
  | < ID: <LETTER> ( <LETTER> | <DIGIT> | "_" )* >
}
```

After the tokens were defined I started to write the grammar based on the grammar definitions defined by the CAL language pdf. After I finished writing out the grammars and fixed some syntax errors, I got warnings that there was 2 left recursion present in the grammar, at the expression function, and the condition function. The problem with the expression function was that expression -> fragment -> expression. I followed this website: "https://www.engr.mun.ca/~theo/Misc/exp_parsing.htm" to figure out how to implement the left recursion rule in javacc. To fix this I split up the expression function into the fragment function so that we could easier see the alpha and beta and eliminate the left recursion. After the function was split there was a choice conflict where I needed to add a lookahead of 2 as the production rule was non-terminal. I also did the same with condition, I split it up but I needed to add a expanded terminal function. So my two functions looked like this:

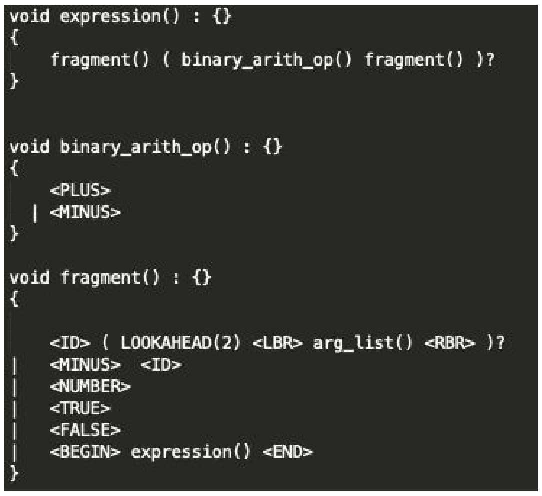


```
void condition() : {}
{
    condition_simple() condition_choice()
}

void condition_choice() : {}
{
    <AND> condition()
|   <OR> condition()
|   {}
}

void condition_simple() : {}
{
    <NOT> condition()
|   LOOKAHEAD(2) <LBR> condition() <RBR>
|   expression() comp_op() expression()
}

void comp_op() : {}
{
    <EQUAL>
|   <NOT_EQUAL>
|   <LESS_THAN>
|   <LESS_THAN_EQUAL_TO>
|   <GREATER_THAN>
|   <GREATER_THAN_EQUAL_TO>
}
```



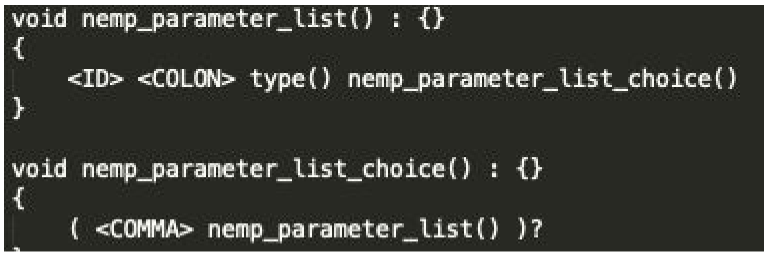
```
void expression() : {}
{
    fragment() ( binary_arith_op() fragment() )?
}


void binary_arith_op() : {}
{
    <PLUS>
  | <MINUS>
}

void fragment() : {}
{

    <ID> ( LOOKAHEAD(2) <LBR> arg_list() <RBR> )?
|   <MINUS>  <ID>
|   <NUMBER>
|   <TRUE>
|   <FALSE>
|   <BEGIN> expression() <END>
}
```

After the left recursion was resolved, I had several choice conflicts caused by ambiguity in the grammar. This was fixed by splitting the functions up to remove the ambiguity. For example in condition when the parser does not know whether to use ‘and’ or ‘or’. Another example of what the functions look like after they are split up is shown.



```
void nemp_parameter_list() : {}
{
    <ID> <COLON> type() nemp_parameter_list_choice()
}

void nemp_parameter_list_choice() : {}
{
    ( <COMMA> nemp_parameter_list() )?
```

The file is run by first compiling the .jj file called ‘cal.jj’ with javacc. After this is done, you must then compiler the .java files with javac. Then you can run the Cal parser with the following command, ‘java CALParser’. You can also feed in a file for the parser to parse using the ‘<’ token.