Assignment 2

Section 1 - Turning Grammar into LL1 Grammar (Non-Ambiguous)

Original Grammar

```
<START>  ::= {{<classDeclOrFuncDef>}}

<classDeclOrFuncDef> ::= <classDecl> | <funcDef>

<classDecl> ::= 'class' 'id' [['isa' 'id' {{',' 'id'}}]] '{'
   {{<visibility> <memberDecl>}} '}' ';'

<visibility>  ::= 'public' | 'private' | EPSILON

<memberDecl> ::= <memberFuncDecl> |<memberVarDecl>

<memberFuncDecl> ::= 'function' 'id' ':' '(' <fParams> ')'
   'arrow' <returnType> ';' | 'constructor' ':' '(' <fParams>
   ')' ';'

<memberVarDecl> ::= 'attribute' 'id' ':' <type> {{<arraySize>}}
   ';'

<funcDef> ::= <funcHead> <funcBody>

<funcHead>  ::= 'function' [['id' 'sr']] 'id' '(' <fParams> ')'
   'arrow' <returnType>
| 'function' 'id' 'sr' 'constructor' '(' <fParams> ')'

<funcBody>  ::= '{' {{<localVarDeclOrStmt>}} '}'

<localVarDeclOrStmt> ::= <localVarDecl> | <statement>

<localVarDecl> ::= 'localVar' 'id' ':' <type> {{<arraySize>}}
   ';'
| 'localVar' 'id' ':' <type> '(' <aParams> ')' ';'

<statement>  ::= <assignStat> ';'
| 'if'  '(' <relExpr> ')' 'then' <statBlock> 'else' <statBlock>
   ';'
| 'while' '(' <relExpr> ')' <statBlock> ';'
| 'read' '(' <variable> ')' ';'
| 'write' '(' <expr> ')' ';'
```

```
| 'return' '(' <expr> ')' ';'
| <functionCall> ';'

<assignStat>  ::= <variable> <assignOp> <expr>

<statBlock> ::= '{' {{<statement>}} '}' | <statement> | EPSILON

<expr>  ::= <arithExpr> | <relExpr>

<relExpr>  ::= <arithExpr> <relOp> <arithExpr>

<arithExpr>  ::= <arithExpr> <addOp> <term> | <term>

<sign>  ::= '+' | '-'

<term> ::= <term> <multOp> <factor> | <factor>

<factor> ::= <variable>
| <functionCall>
| 'intLit' | 'floatLit'
| '(' <arithExpr> ')'
| 'not' <factor>
| <sign> <factor>

<variable> ::= {{<idnest>}} 'id' {{<indice>}}

<functionCall> ::= {{<idnest>}} 'id' '(' <aParams> ')'

<idnest> ::= 'id' {{<indice>}} '.'
| 'id' '(' <aParams> ')' '.'

<indice> ::= '[' <arithExpr> ']'

<arraySize> ::= '[' 'intLit' ']' | '[' ']'

<type> ::= 'integer' | 'float' | 'id'

<returnType> ::= <type> | 'void'

<fParams> ::= 'id' ':' <type> {{<arraySize>}} {{<fParamsTail>}}
   | EPSILON

<aParams> ::= <expr> {{<aParamsTail>}} | EPSILON

<fParamsTail>  ::= ',' 'id' ':' <type> {{<arraySize>}}

<aParamsTail>  ::= ',' <expr>
```

```
<assignOp> ::= '='

<relOp> ::= 'eq' | 'neq' | 'lt' | 'gt' | 'leq' | 'geq'

<addOp> ::= '+' | '-' | 'or'

<multOp> ::= '*' | '/' | 'and’
```

Grammar after using *grammartool.jar* by Joey Paquet (Left Recursion, EBNF Optionality, EBNF Repetition removed)

```
<START> ::= <rept-START0>

<aParams> ::= <expr> <rept-aParams1>
<aParams> ::= EPSILON

<aParamsTail> ::= ',' <expr>

<addOp> ::= '+'
<addOp> ::= '-'
<addOp> ::= 'or'

<arithExpr> ::= <term> <rightrec-arithExpr>

<arraySize> ::= '[' 'intLit' ']'
<arraySize> ::= '[' ']'

<assignOp> ::= '='

<assignStat> ::= <variable> <assignOp> <expr>

<classDecl> ::= 'class' 'id' <opt-classDecl2> '{'
   <rept-classDecl4> '}' ';'

<classDeclOrFuncDef> ::= <classDecl>
<classDeclOrFuncDef> ::= <funcDef>

<expr> ::= <arithExpr>
<expr> ::= <relExpr>

<fParams> ::= 'id' ':' <type> <rept-fParams3> <rept-fParams4>
<fParams> ::= EPSILON

<fParamsTail> ::= ',' 'id' ':' <type> <rept-fParamsTail4>

<factor> ::= <variable>
<factor> ::= <functionCall>
<factor> ::= 'intLit'
```

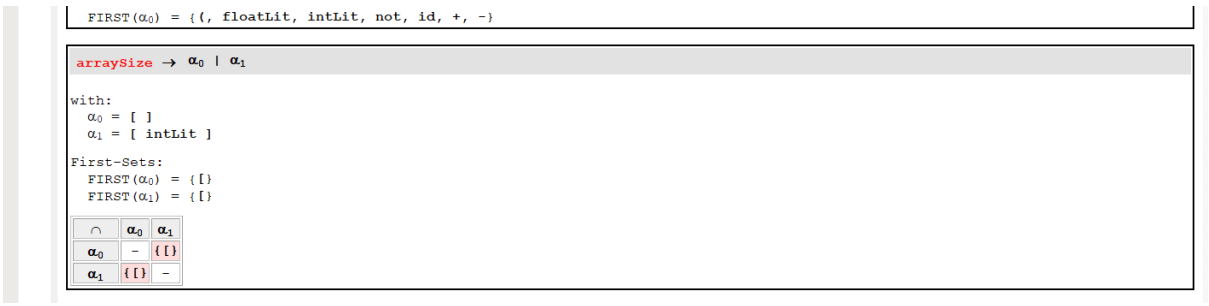
```
<factor> ::= 'floatLit'
<factor> ::= '(' <arithExpr> ')'
<factor> ::= 'not' <factor>
<factor> ::= <sign> <factor>

<funcBody> ::= '{' <rept-funcBody1> '}'

<funcDef> ::= <funcHead> <funcBody>

<funcHead> ::= 'function' <opt-funcHead1> 'id' '(' <fParams> ')'
   'arrow' <returnType>
<funcHead> ::= 'function' 'id' 'sr' 'constructor' '(' <fParams>
   ')'

<functionCall> ::= <rept-functionCall0> 'id' '(' <aParams> ')'

<idnest> ::= 'id' <rept-idnest1> '.'
<idnest> ::= 'id' '(' <aParams> ')' '.'

<indice> ::= '[' <arithExpr> ']'

<localVarDecl> ::= 'localVar' 'id' ':' <type>
   <rept-localVarDecl4> ';'
<localVarDecl> ::= 'localVar' 'id' ':' <type> '(' <aParams> ')'
   ';'

<localVarDeclOrStmt> ::= <localVarDecl>
<localVarDeclOrStmt> ::= <statement>

<memberDecl> ::= <memberFuncDecl>
<memberDecl> ::= <memberVarDecl>

<memberFuncDecl> ::= 'function' 'id' ':' '(' <fParams> ')'
   'arrow' <returnType> ';'
<memberFuncDecl> ::= 'constructor' ':' '(' <fParams> ')' ';'

<memberVarDecl> ::= 'attribute' 'id' ':' <type>
   <rept-memberVarDecl4> ';'

<multOp> ::= '*'
<multOp> ::= '/'
<multOp> ::= 'and'

<opt-classDecl2> ::= 'isa' 'id' <rept-opt-classDecl22>
<opt-classDecl2> ::= EPSILON

<opt-funcHead1> ::= 'id' 'sr'
<opt-funcHead1> ::= EPSILON
```

```
<relExpr> ::= <arithExpr> <relOp> <arithExpr>

<relOp> ::= 'eq'
<relOp> ::= 'neq'
<relOp> ::= 'lt'
<relOp> ::= 'gt'
<relOp> ::= 'leq'
<relOp> ::= 'geq'

<rept-START0> ::= <classDeclOrFuncDef> <rept-START0>
<rept-START0> ::= EPSILON

<rept-aParams1> ::= <aParamsTail> <rept-aParams1>
<rept-aParams1> ::= EPSILON

<rept-classDecl4> ::= <visibility> <memberDecl>
   <rept-classDecl4>
<rept-classDecl4> ::= EPSILON

<rept-fParams3> ::= <arraySize> <rept-fParams3>
<rept-fParams3> ::= EPSILON

<rept-fParams4> ::= <fParamsTail> <rept-fParams4>
<rept-fParams4> ::= EPSILON

<rept-fParamsTail4> ::= <arraySize> <rept-fParamsTail4>
<rept-fParamsTail4> ::= EPSILON

<rept-funcBody1> ::= <localVarDeclOrStmt> <rept-funcBody1>
<rept-funcBody1> ::= EPSILON

<rept-functionCall0> ::= <idnest> <rept-functionCall0>
<rept-functionCall0> ::= EPSILON

<rept-idnest1> ::= <indice> <rept-idnest1>
<rept-idnest1> ::= EPSILON

<rept-localVarDecl4> ::= <arraySize> <rept-localVarDecl4>
<rept-localVarDecl4> ::= EPSILON

<rept-memberVarDecl4> ::= <arraySize> <rept-memberVarDecl4>
<rept-memberVarDecl4> ::= EPSILON

<rept-opt-classDecl22> ::= ',' 'id' <rept-opt-classDecl22>
<rept-opt-classDecl22> ::= EPSILON

<rept-statBlock1> ::= <statement> <rept-statBlock1>
```

```
<rept-statBlock1> ::= EPSILON

<rept-variable0> ::= <idnest> <rept-variable0>
<rept-variable0> ::= EPSILON

<rept-variable2> ::= <indice> <rept-variable2>
<rept-variable2> ::= EPSILON

<returnType> ::= <type>
<returnType> ::= 'void'

<rightrec-arithExpr> ::= EPSILON
<rightrec-arithExpr> ::= <addOp> <term> <rightrec-arithExpr>

<rightrec-term> ::= EPSILON
<rightrec-term> ::= <multOp> <factor> <rightrec-term>

<sign> ::= '+'
<sign> ::= '-'

<statBlock> ::= '{' <rept-statBlock1> '}'
<statBlock> ::= <statement>
<statBlock> ::= EPSILON

<statement> ::= <assignStat> ';'
<statement> ::= 'if' '(' <relExpr> ')' 'then' <statBlock> 'else'
   <statBlock> ';'
<statement> ::= 'while' '(' <relExpr> ')' <statBlock> ';'
<statement> ::= 'read' '(' <variable> ')' ';'
<statement> ::= 'write' '(' <expr> ')' ';'
<statement> ::= 'return' '(' <expr> ')' ';'
<statement> ::= <functionCall> ';'

<term> ::= <factor> <rightrec-term>

<type> ::= 'integer'
<type> ::= 'float'
<type> ::= 'id'

<variable> ::= <rept-variable0> 'id' <rept-variable2>

<visibility> ::= 'public'
<visibility> ::= 'private'
<visibility> ::= EPSILON
```

Ambiguities found using [kfG Edit] - Fixed using [CFG to LL(k) (cyberzhg.github.io)] and by manual edits

1.) arraySize



Original:

```
arraySize -> '[' 'intLit' ']'
arraySize -> '[' ']'
```

Fixed:

```
arraySize -> '[' arraySize'
arraySizePrime -> intLit ']'
arraySizePrime -> ']'
```

2.) funcHead

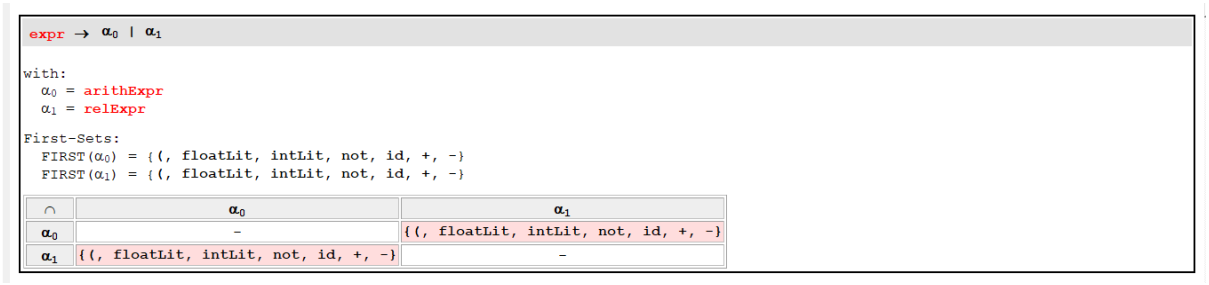
Original:

```
funcHead -> 'function' opt-funcHead1 'id' '(' fParams ')'
   'arrow' returnType
funcHead -> 'function' 'id' 'sr' 'constructor' '(' fParams ')'

opt-funcHead1 -> 'id' 'sr'
opt-funcHead1 -> EPSILON
```

Fixed:

```
funcHead -> 'function' 'id' funcHeadPrime

funcHeadPrime -> '(' fParams ')' 'arrow' returnType
funcHeadPrime -> 'sr' funcHeadPrimePrime

funcHeadPrimePrime -> 'constructor' '(' fParams ')'
funcHeadPrimePrime -> 'id' '(' fParams ')' 'arrow' returnType
```

3.) expr



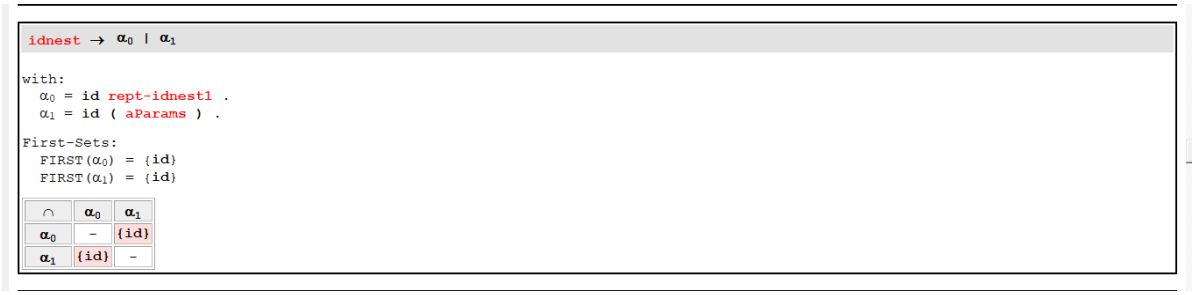
Original:

```
expr -> arithExpr
expr -> relExpr

arithExpr -> term rightrec-arithExpr

term -> factor rightrec-term

relExpr -> arithExpr relOp arithExpr
```

Fixed:

```
expr -> term exprEnd

exprEnd -> arithExprEnd
exprEnd -> rightrec-arithExpr

arithExpr -> term arithExprEnd

arithExprEnd - > rightrec-arithExpr

relExpr -> term relExprEnd

relExprEnd -> rightrec-arithExpr relOp arithExpr

term -> factor rightrec-term
```

4.) idnest



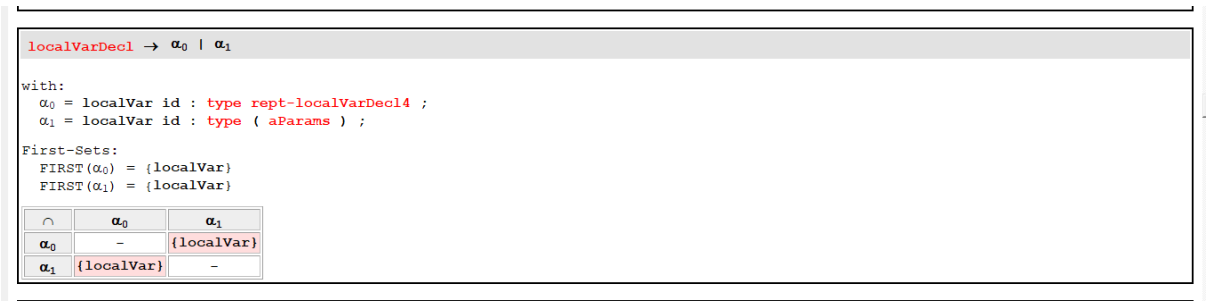
Original:

```
idnest -> 'id' rept-idnest1 '.'
idnest -> 'id' '(' aParams ')' '.'
```

Fixed:

```
idnest -> 'id' idnestPrime

idnestPrime -> rept-idnest1 '.'
idnestPrime -> '(' aParams ')' '.'
```

5.) localVarDecl



Original:

```
localVarDecl -> 'localVar' 'id' ':' type rept-localVarDecl4 ';'
localVarDecl -> 'localVar' 'id' ':' type '(' aParams ')' ';'
```

Fixed:

```
localVarDecl -> 'localVar' 'id' ':' type localVarDeclPrime

localVarDeclPrime -> rept-localVarDecl4 ';'
localVarDeclPrime -> '(' aParams ')' ';'
```

6.) variable

There was intersect between the first sets of variable, and rept-varOrFunc that needed to be fixed.

Original:

```
variable -> rept-varOrFunc 'id' variableEnd
variableEnd -> rept-variable2

rept-varOrFunc -> idnest rept-varOrFunc
rept-varOrFunc -> EPSILON

rept-variable2 -> indice rept-variable2
rept-variable2 -> EPSILON

idnest -> 'id' idnestPrime

idnestPrime -> rept-idnest1 '.'
idnestPrime -> '(' aParams ')' '.'
```

Fixed:

```
variable -> 'id' variablePrime

variablePrime -> rept-idnest variablePrimePrime
variablePrime -> '(' aParams ')' '.' variable

variablePrimePrime -> '.' variable
variablePrimePrime -> EPSILON
```

7.) function

There was intersect between functionCall and rept-functionCall0 that needed to be fixed.

Original:

```
<functionCall> ::= <rept-functionCall0> 'id' '(' <aParams> ')'

rept-functionCall0 -> idnest rept-functionCall0
rept-functionCall0 -> EPSILON
```

Fixed:

```
functionCall -> 'id' functionCallPrime

functionCallPrime -> '(' aParams ')' functionPrimePrime
functionCallPrime -> rept-idnest1 '.' variable

functionPrimePrime -> '.' function
functionPrimePrime -> EPSILON
```

8.) factor

factor → α_0 | α_1 | α_2 | α_3 | α_4 | α_5 | α_6

with:
- α_0 = sign factor
- α_1 = functionCall
- α_2 = variable
- α_3 = not factor
- α_4 = intLit
- α_5 = floatLit
- α_6 = (arithExpr)

First-Sets:
- FIRST(α_0) = {+, -}
- FIRST(α_1) = {id}
- FIRST(α_2) = {id}
- FIRST(α_3) = {not}
- FIRST(α_4) = {intLit}
- FIRST(α_5) = {floatLit}
- FIRST(α_6) = {(}

∩	α_0	α_1	α_2	α_3	α_4	α_5	α_6
α_0	-	∅	∅	∅	∅	∅	∅
α_1	∅	-	{id}	∅	∅	∅	∅
α_2	∅	{id}	-	∅	∅	∅	∅
α_3	∅	∅	∅	-	∅	∅	∅
α_4	∅	∅	∅	∅	-	∅	∅
α_5	∅	∅	∅	∅	∅	-	∅
α_6	∅	∅	∅	∅	∅	∅	-

Original:

```
factor -> variable
factor -> functionCall
factor -> 'intLit'
factor -> 'floatLit'
factor -> '(' arithExpr ')'
factor -> 'not' factor
factor -> sign factor

variable -> rept-variable0 'id' rept-variable2

rept-variable0 -> idnest rept-variable0
rept-variable0 -> EPSILON

functionCall -> rept-functionCall0 'id' '(' aParams ')'

rept-functionCall0 -> idnest rept-functionCall0
rept-functionCall0 -> EPSILON
```

Fixed:

```
factor -> variableOrFunctionCall
factor -> 'intLit'
factor -> 'floatLit'
factor -> '(' arithExpr ')'
factor -> 'not' factor
factor -> sign factor

variableOrFunctionCall -> 'id' variableOrFunctionCallPrime

variableOrFunctionCallPrime -> indice
   variableOrFunctionCallPrimePrime
variableOrFunctionCallPrime -> '(' aParams ')'
   variableOrFunctionCallPrimePrime
variableOrFunctionCallPrime -> variableOrFunctionCallPrimePrime
variableOrFunctionCallPrime -> EPSILON

variableOrFunctionCallPrimePrime -> '.' variableOrFunctionCall
variableOrFunctionCallPrimePrime -> EPSILON
```

9.) statement

statement → α_0 | α_1 | α_2 | α_3 | α_4 | α_5 | α_6

with:
- α_0 = assignStat ;
- α_1 = functionCall ;
- α_2 = write (expr) ;
- α_3 = while (relExpr) statBlock ;
- α_4 = return (expr) ;
- α_5 = read (variable) ;
- α_6 = if (relExpr) then statBlock else statBlock ;

First-Sets:
- FIRST(α_0) = {id}
- FIRST(α_1) = {id}
- FIRST(α_2) = {write}
- FIRST(α_3) = {while}
- FIRST(α_4) = {return}
- FIRST(α_5) = {read}
- FIRST(α_6) = {if}

∩	α_0	α_1	α_2	α_3	α_4	α_5	α_6
α_0	-	{id}	∅	∅	∅	∅	∅
α_1	{id}	-	∅	∅	∅	∅	∅
α_2	∅	∅	-	∅	∅	∅	∅
α_3	∅	∅	∅	-	∅	∅	∅
α_4	∅	∅	∅	∅	-	∅	∅
α_5	∅	∅	∅	∅	∅	-	∅
α_6	∅	∅	∅	∅	∅	∅	-

Original:

```
statement -> assignStat ';'
statement -> 'if' '(' relExpr ')' 'then' statBlock 'else'
   statBlock ';'
statement -> 'while' '(' relExpr ')' statBlock ';'
statement -> 'read' '(' variable ')' ';'
statement -> 'write' '(' expr ')' ';'
statement -> 'return' '(' expr ')' ';'
statement -> functionCall ';'

assignStat -> variable assignOp expr

variableOrFuncEnd -> variableEnd
variableOrFuncEnd -> functionCallEnd

variable -> rept-varOrFunc  'id'  variableEnd
variableEnd -> rept-variable2

rept-varOrFunc -> idnest rept-varOrFunc
rept-variable0 -> EPSILON

functionCall -> rept-varOrFunc  'id'  functionCallEnd
functionCallEnd -> '(' aParams ')'
```

Fixed:

```
statement -> assignStatOrFunctionCall ';'
statement -> 'if' '(' relExpr ')' 'then' statBlock 'else'
   statBlock ';'
statement -> 'while' '(' relExpr ')' statBlock ';'
statement -> 'read' '(' variable ')' ';'
statement -> 'write' '(' expr ')' ';'
statement -> 'return' '(' expr ')' ';'

assignStatOrFunctionCall -> 'id' assignStatOrFunctionCallPrime

assignStatOrFunctionCallPrime -> '(' aParams ')'
   assignStatOrFunctionCallPrime
assignStatOrFunctionCallPrime -> rept-idnest1
   assignStatOrFunctionCallPrimePrime

assignStatOrFunctionCallPrimePrime -> '.'
   assignStatOrFunctionCallPrime
assignStatOrFunctionCallPrimePrime -> EPSILON

assignStatOrFunctionCallPrimePrimePrime -> '.'
   assignStatOrFunctionCallPrime
assignStatOrFunctionCallPrimePrimePrime -> assignOp expr
```

Other Changes

- Had to add this to the grammar cause ‘self’ was being broken
    - assignStatOrFunctionCall -> 'self' assignStatOrFunctionCallPrime
- I changed ‘localVar’ to ‘localvar’ in the grammar so it would match with the given source files,
    - Did similar thing to “sr”
    - Also changed “intLit” and “floatLit” to “intnum” and “floatnum” to match the tokens

Final grammar:

```
START -> rept-START0

aParams -> expr rept-aParams1
aParams -> EPSILON

aParamsTail -> ',' expr

addOp -> '+'
addOp -> '-'
addOp -> 'or'

arraySize -> '[' arraySizePrime
arraySizePrime -> 'intnum' ']'
arraySizePrime -> ']'

assignOp -> '='

classDecl -> 'class' 'id' opt-classDecl2 '{' rept-classDecl4 '}'
   ';'

classDeclOrFuncDef -> classDecl
classDeclOrFuncDef -> funcDef

expr -> term rightrec-arithExpr

arithExpr -> term rightrec-arithExpr

relExpr -> term relExprEnd

relExprEnd -> rightrec-arithExpr relOp arithExpr

term -> factor rightrec-term

factor -> variableOrFunctionCall
factor -> 'intnum'
factor -> 'floatnum'
factor -> '(' arithExpr ')'
factor -> 'not' factor
factor -> sign factor

rightrec-arithExpr -> EPSILON
rightrec-arithExpr -> addOp term rightrec-arithExpr

fParams -> 'id' ':' type rept-fParams3 rept-fParams4
fParams -> EPSILON
```

```
fParamsTail -> ',' 'id' ':' type rept-fParamsTail4

funcBody -> '{' rept-funcBody1 '}'

funcDef -> funcHead funcBody

funcHead -> 'function' 'id' funcHeadPrime

funcHeadPrime -> '(' fParams ')' 'arrow' returnType
funcHeadPrime -> 'sr' funcHeadPrimePrime

funcHeadPrimePrime -> 'constructor' '(' fParams ')'
funcHeadPrimePrime -> 'id' '(' fParams ')' 'arrow' returnType

indice -> '[' arithExpr ']'

localVarDecl -> 'localvar' 'id' ':' type localVarDeclPrime

localVarDeclPrime -> rept-localVarDecl4 ';'
localVarDeclPrime -> '(' aParams ')' ';'

localVarDeclOrStmt -> localVarDecl
localVarDeclOrStmt -> statement

memberDecl -> memberFuncDecl
memberDecl -> memberVarDecl

memberFuncDecl -> 'function' 'id' ':' '(' fParams ')' 'arrow'
   returnType ';'
memberFuncDecl -> 'constructor' ':' '(' fParams ')' ';'

memberVarDecl -> 'attribute' 'id' ':' type rept-memberVarDecl4
   ';'

multOp -> '*'
multOp -> '/'
multOp -> 'and'

opt-classDecl2 -> 'isa' 'id' rept-opt-classDecl22
opt-classDecl2 -> EPSILON

relOp -> 'eq'
relOp -> 'neq'
relOp -> 'lt'
relOp -> 'gt'
relOp -> 'leq'
relOp -> 'geq'
```

```
rept-START0 -> classDeclOrFuncDef rept-START0
rept-START0 -> EPSILON

rept-aParams1 -> aParamsTail rept-aParams1
rept-aParams1 -> EPSILON

rept-classDecl4 -> visibility memberDecl rept-classDecl4
rept-classDecl4 -> EPSILON

rept-fParams3 -> arraySize rept-fParams3
rept-fParams3 -> EPSILON

rept-fParams4 -> fParamsTail rept-fParams4
rept-fParams4 -> EPSILON

rept-fParamsTail4 -> arraySize rept-fParamsTail4
rept-fParamsTail4 -> EPSILON

rept-funcBody1 -> localVarDeclOrStmt rept-funcBody1
rept-funcBody1 -> EPSILON

rept-idnest1 -> indice rept-idnest1
rept-idnest1 -> EPSILON

rept-localVarDecl4 -> arraySize rept-localVarDecl4
rept-localVarDecl4 -> EPSILON

rept-memberVarDecl4 -> arraySize rept-memberVarDecl4
rept-memberVarDecl4 -> EPSILON

rept-opt-classDecl22 -> ',' 'id' rept-opt-classDecl22
rept-opt-classDecl22 -> EPSILON

rept-statBlock1 -> statement rept-statBlock1
rept-statBlock1 -> EPSILON

returnType -> type
returnType -> 'void'

sign -> '+'
sign -> '-'

statBlock -> '{' rept-statBlock1 '}'
statBlock -> statement
statBlock -> EPSILON

statement -> assignStatOrFunctionCall ';'
```

```
statement -> 'if' '(' relExpr ')' 'then' statBlock 'else'
   statBlock ';'
statement -> 'while' '(' relExpr ')' statBlock ';'
statement -> 'read' '(' variable ')' ';'
statement -> 'write' '(' expr ')' ';'
statement -> 'return' '(' expr ')' ';'

assignStat -> variable assignOp expr

variable -> 'id' variablePrime

variablePrime -> rept-idnest1 variablePrimePrime
variablePrime -> '(' aParams ')' '.' variable

variablePrimePrime -> '.' variable
variablePrimePrime -> EPSILON

functionCall -> 'id' functionCallPrime

functionCallPrime -> '(' aParams ')' functionPrimePrime
functionCallPrime -> rept-idnest1 '.' variable

functionPrimePrime -> '.' function
functionPrimePrime -> EPSILON

variableOrFunctionCall -> 'id' variableOrFunctionCallPrime

variableOrFunctionCallPrime -> indice
   variableOrFunctionCallPrimePrime
variableOrFunctionCallPrime -> '(' aParams ')'
   variableOrFunctionCallPrimePrime
variableOrFunctionCallPrime -> variableOrFunctionCallPrimePrime
variableOrFunctionCallPrime -> EPSILON

variableOrFunctionCallPrimePrime -> '.' variableOrFunctionCall
variableOrFunctionCallPrimePrime -> EPSILON

assignStatOrFunctionCall -> 'id' assignStatOrFunctionCallPrime
assignStatOrFunctionCall -> 'self' assignStatOrFunctionCallPrime

assignStatOrFunctionCallPrime -> '(' aParams ')'
   assignStatOrFunctionCallPrimePrime
assignStatOrFunctionCallPrime -> rept-idnest1
   assignStatOrFunctionCallPrimePrimePrime

assignStatOrFunctionCallPrimePrime -> '.'
   assignStatOrFunctionCall
assignStatOrFunctionCallPrimePrime -> EPSILON
```

```
assignStatOrFunctionCallPrimePrimePrime -> '.'
   assignStatOrFunctionCall
assignStatOrFunctionCallPrimePrimePrime -> assignOp expr

idnest -> 'id' idnestPrime

idnestPrime -> rept-idnest1 '.'
idnestPrime -> '(' aParams ')' '.'

rept-idnest -> idnest rept-idnest
rept-idnest -> EPSILON

rightrec-term -> EPSILON
rightrec-term -> multOp factor rightrec-term

type -> 'integer'
type -> 'float'
type -> 'id'

visibility -> 'public'
visibility -> 'private'
visibility -> EPSILON
```

Section 2 - First and Follow Sets

The first and follow sets are generated in the code from the grammar. To see this code look inside GrammarTableGenerator.generateGrammarTable(). They can be printed using the method GrammarTableGenerator.printFirstAndFollowSets().

First Sets

fParamsTail: ,,
variablePrime: (, [, .,
localVarDeclPrime: (, [, ;,
sign: +, -,
aParams: not, intnum, (, floatnum, id, +, -,
classDeclOrFuncDef: function, class,
type: id, integer, float,
rept-funcBody1: localvar, read, self, id, while, write, if, return,
arithExpr: not, intnum, (, floatnum, id, +, -,
multOp: and, *, /,
idnest: id,
assignStatOrFunctionCallPrimePrimePrime: =, .,
rept-idnest1: [,

variableOrFunctionCall: id,
localVarDeclOrStmt: localvar, read, self, id, while, write, if, return,
memberFuncDecl: function, constructor,
memberVarDecl: attribute,
functionPrimePrime: .,
rept-opt-classDecl22: ,,
rept-classDecl4: private, public, function, constructor, attribute,
visibility: private, public,
indice: [,
classDecl: class,
relExprEnd: geq, or, lt, leq, +, neq, eq, -, gt,
variableOrFunctionCallPrimePrime: .,
relOp: geq, lt, leq, neq, eq, gt,
variableOrFunctionCallPrime: (, [, .,
rept-idnest: id,
arraySizePrime: intnum,],
assignStatOrFunctionCallPrimePrime: .,
localVarDecl: localvar,
rept-memberVarDecl4: [,
functionCall: id,
funcBody: {,
returnType: void, id, integer, float,
funcDef: function,
aParamsTail: ,,
funcHeadPrime: (, sr,
assignOp: =,
fParams: id,
rept-fParams3: [,
rept-fParams4: ,,
statBlock: read, self, id, {, while, write, if, return,
opt-classDecl2: isa,
functionCallPrime: (, [, .,
statement: read, self, id, while, write, if, return,
expr: not, intnum, (, floatnum, id, +, -,
term: not, intnum, (, floatnum, id, +, -,
factor: not, intnum, (, floatnum, id, +, -,
relExpr: not, intnum, (, floatnum, id, +, -,
funcHeadPrimePrime: constructor, id,
rept-localVarDecl4: [,
rightrec-arithExpr: or, +, -,
rept-fParamsTail4: [,
addOp: or, +, -,
rept-START0: function, class,
rightrec-term: and, *, /,
arraySize: [,
rept-aParams1: ,,
rept-statBlock1: read, self, id, while, write, if, return,
memberDecl: function, constructor, attribute,

variablePrimePrime: .,
assignStatOrFunctionCall: self, id,
assignStatOrFunctionCallPrime: (, [, =, .,
variable: id,
START: function, class,
idnestPrime: (, [, .,
assignStat: id,
funcHead: function,

FOLLOW SETS:

fParamsTail: ')' ','
variablePrime: ')' '='
localVarDeclPrime: 'localvar' 'read' 'self' 'id' 'while' 'write' 'if' '}' 'return'
sign: 'not' 'intnum' '(' 'floatnum' 'id' '+' '-'
aParams: ')'
classDeclOrFuncDef: '$' 'function' 'class'
type: '(' ')' '[' ';' '{' ','
rept-funcBody1: '}'
arithExpr: ')' ']'
multOp: 'not' 'intnum' '(' 'floatnum' 'id' '+' '-'
idnest: 'id'
assignStatOrFunctionCallPrimePrimePrime: ';'
rept-idnest1: ')' '=' '.'
localVarDeclOrStmt: 'localvar' 'read' 'self' 'id' 'while' 'write' 'if' '}' 'return'
variableOrFunctionCall: 'or' 'lt' ')' '*' '+' ',' 'eq' '-' 'gt' '/' 'geq' 'and' 'leq' ';' 'neq' ']'
memberFuncDecl: 'private' 'public' 'function' 'constructor' 'attribute' '}'
memberVarDecl: 'private' 'public' 'function' 'constructor' 'attribute' '}'
functionPrimePrime:
rept-opt-classDecl22: '{'
rept-classDecl4: '}'
visibility: 'function' 'constructor' 'attribute'
indice: 'or' 'lt' ')' '*' '+' ',' 'eq' '-' 'gt' '.' '/' 'geq' 'and' 'leq' '[' ';' 'neq' '=' ']'
classDecl: '$' 'function' 'class'
relExprEnd: ')'
variableOrFunctionCallPrimePrime: 'or' 'lt' ')' '*' '+' ',' 'eq' '-' 'gt' '/' 'geq' 'and' 'leq' ';' 'neq' ']'
variableOrFunctionCallPrime: 'or' 'lt' ')' '*' '+' ',' 'eq' '-' 'gt' '/' 'geq' 'and' 'leq' ';' 'neq' ']'
relOp: 'not' 'intnum' '(' 'floatnum' 'id' '+' '-'
rept-idnest:
arraySizePrime: ')' '[' ';' ','
assignStatOrFunctionCallPrimePrime: ';'
localVarDecl: 'localvar' 'read' 'self' 'id' 'while' 'write' 'if' '}' 'return'
functionCall:
rept-memberVarDecl4: ';'
returnType: ';' '{'
funcBody: '$' 'function' 'class'

funcHeadPrime: '{'
aParamsTail: ')' ','
funcDef: '$' 'function' 'class'
assignOp: 'not' 'intnum' '(' 'floatnum' 'id' '+' '-'
fParams: ')'
rept-fParams3: ')' ','
rept-fParams4: ')'
statBlock: 'else' ';'
functionCallPrime:
opt-classDecl2: '{'
statement: 'localvar' 'read' 'else' 'self' 'id' ';' 'while' 'write' 'if' '}' 'return'
term: 'or' 'lt' ')' '+' ',' 'eq' '-' 'gt' 'geq' 'leq' ';' 'neq' ']'
expr: ')' ';' ','
relExpr: ')'
factor: 'or' 'lt' ')' '*' '+' ',' 'eq' '-' 'gt' '/' 'geq' 'and' 'leq' ';' 'neq' ']'
funcHeadPrimePrime: '{'
rightrec-arithExpr: 'geq' 'lt' ')' 'leq' 'neq' ';' 'eq' ',' ']' 'gt'
rept-localVarDecl4: ';'
rept-fParamsTail4: ')' ','
rept-START0: '$'
addOp: 'not' 'intnum' '(' 'floatnum' 'id' '+' '-'
rightrec-term: 'or' 'lt' ')' '+' ',' 'eq' '-' 'gt' 'geq' 'leq' ';' 'neq' ']'
rept-aParams1: ')'
arraySize: ')' '[' ';' ','
rept-statBlock1: '}'
memberDecl: 'private' 'public' 'function' 'constructor' 'attribute' '}'
variablePrimePrime: ')' '='
assignStatOrFunctionCall: ';'
assignStatOrFunctionCallPrime: ';'
variable: ')' '='
START: '$'
idnestPrime: 'id'
assignStat:
funcHead: '{'

Section 3 - Design

Execution starts in the Driver class where the main() method is located.

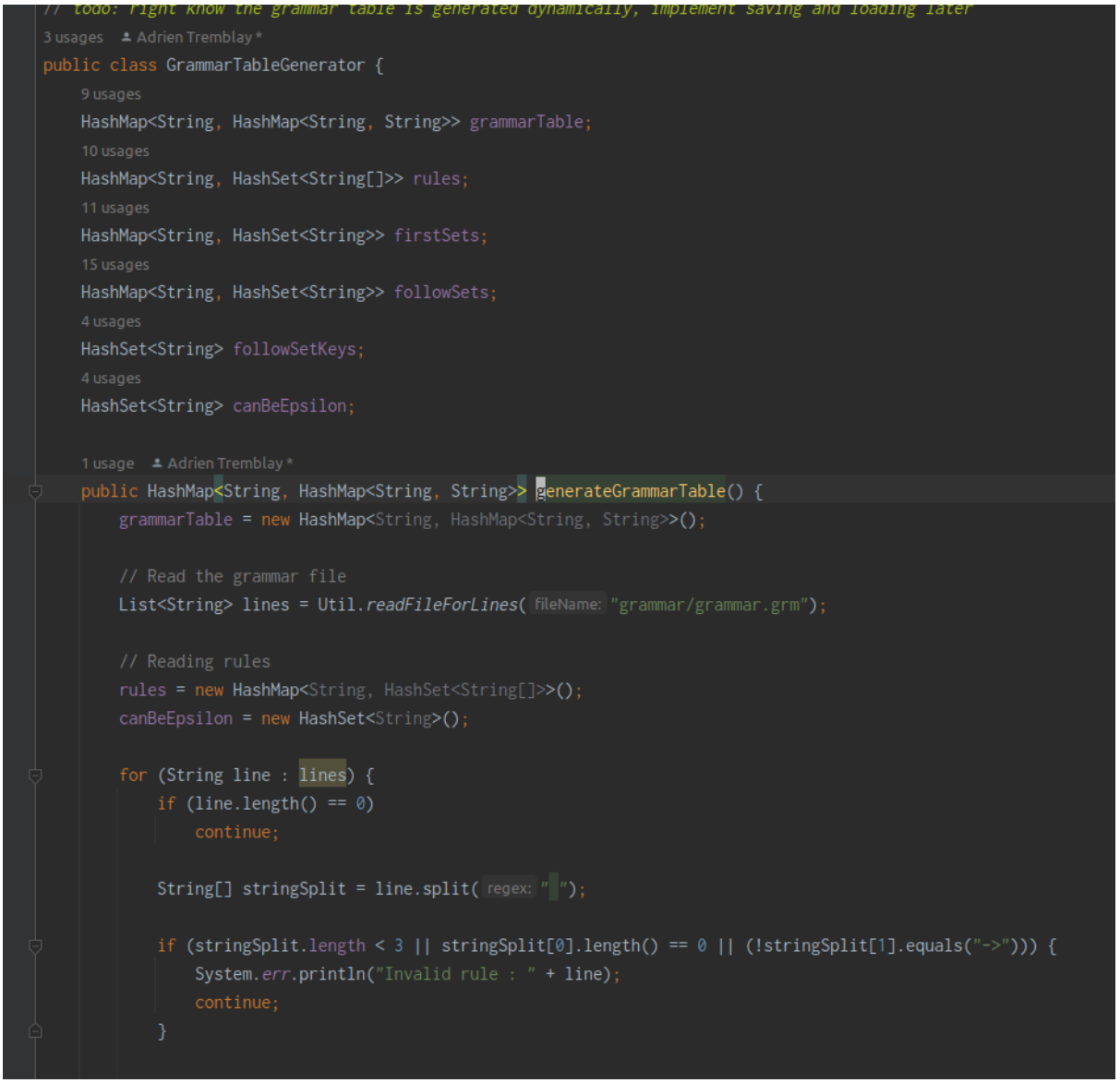
```java
Adrien Tremblay *
public class Driver {

    1 usage
    private static final String[] SOURCE_FILES = {
            //        "test_source_files/my_test/my_test.src",
                      "test_source_files/bubble_sort/bubble_sort.src",
            //        "test_source_files/lex_negative_grading/lex_negative_grading.src",
            //        "test_source_files/lex_positive_grading/lex_positive_grading.src",
            //        "test_source_files/polynomial/polynomial.src"
    };

    Adrien Tremblay
    public static void main(String args[]) throws IOException {
        Parser parser = new Parser();

        for (String sourceFilePath : SOURCE_FILES) {
            parser.loadSource(sourceFilePath);
            parser.parse();
        }
    }
}
```

A Parser is created. Then the parser is loaded with different source files and instructed to parse each file.

```java
3 usages  Adrien Tremblay
public class Parser {
    5 usages
    private Lexer lexer;

    8 usages
    private HashMap<String, HashMap<String, String>> grammarTable;
    2 usages
    private SyntaxDerivationPrinter syntaxDerivationPrinter;


    1 usage  Adrien Tremblay
    public Parser() {
        lexer = new Lexer();

        GrammarTableGenerator grammarTableGenerator = new GrammarTableGenerator();
        grammarTable = grammarTableGenerator.generateGrammarTable();
    }
```

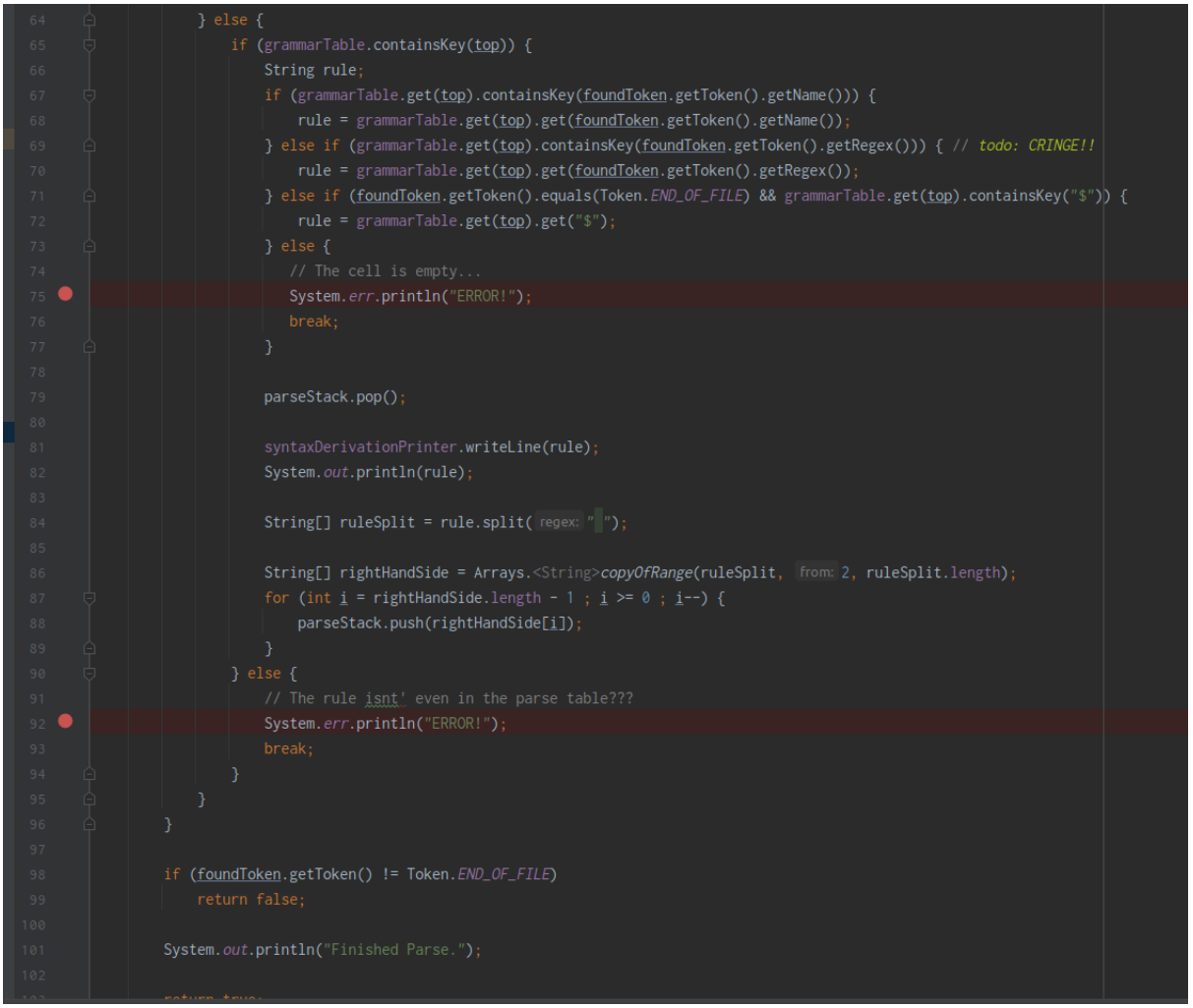
The Parser class maintains a Lexer and a grammarTable. It also owns a SyntaxDerivationPrinter which is a utility class that contains all the code to write to the

derivation file.



A grammar table generator has one primary method generateGrammarTable which generates a grammar table. It maintains a HashMap for the rules, firstSets, and followSets which are used during the generation process. To generate a grammar table, the grammar file is read, then first sets are generated, then the follow sets. Then this information is combined to form the grammar table which takes the form of a hashmap of hashmaps. This code is quite long and complicated but I can go into detail during the demo.

```
1 usage   Adrien Tremblay
public boolean parse() {
    System.out.println("Starting Parse.");

    Stack<String> parseStack = new Stack<String>();

    parseStack.push( item: "$");
    parseStack.push( item: "START");

    FoundToken foundToken = lexer.nextToken();
    String top;
    while ((top = parseStack.peek()) != "$") {
        // skips
        if (foundToken.getToken() == Token.IN_LINE_COMMENT || foundToken.getToken() == Token.BLOCK_COMMENT) {
            foundToken = lexer.nextToken();
            continue;
        }

        if (top.equals("EPSILON")) {
            parseStack.pop();
            continue;
        }

        if (GrammarTableGenerator.isTerminal(top)) {
            String topTerminal = top.substring(1, top.length() - 1);
            if (
                topTerminal.equals(foundToken.getToken().getName()) // for reserved words
                || topTerminal.equals(foundToken.getToken().getRegex()) // for everything else
                || (topTerminal.equals("id") && foundToken.getToken().isType()) // type tokens should count as identifiers too
            ) {
                // found a terminal
                System.out.println("DEBUG: FOUND " + foundToken.getLexeme());
                parseStack.pop();
                foundToken = lexer.nextToken();
            } else {
                // did not find the terminal I wanted to... :'(
                System.err.println("ERROR!");
                break;
            }
```



```
        } else {
            if (grammarTable.containsKey(top)) {
                String rule;
                if (grammarTable.get(top).containsKey(foundToken.getToken().getName())) {
                    rule = grammarTable.get(top).get(foundToken.getToken().getName());
                } else if (grammarTable.get(top).containsKey(foundToken.getToken().getRegex())) { // todo: CRINGE!!
                    rule = grammarTable.get(top).get(foundToken.getToken().getRegex());
                } else if (foundToken.getToken().equals(Token.END_OF_FILE) && grammarTable.get(top).containsKey("$")) {
                    rule = grammarTable.get(top).get("$");
                } else {
                   // The cell is empty...
                   System.err.println("ERROR!");
                   break;
                }

                parseStack.pop();

                syntaxDerivationPrinter.writeLine(rule);
                System.out.println(rule);

                String[] ruleSplit = rule.split( regex: " ");

                String[] rightHandSide = Arrays.<String>copyOfRange(ruleSplit, from: 2, ruleSplit.length);
                for (int i = rightHandSide.length - 1 ; i >= 0 ; i--) {
                    parseStack.push(rightHandSide[i]);
                }
            } else {
                // The rule isnt' even in the parse table???
                System.err.println("ERROR!");
                break;
            }
        }
    }

    if (foundToken.getToken() != Token.END_OF_FILE)
        return false;

    System.out.println("Finished Parse.");

    return true;
```

The parsing logic is located in the Parser.parse() method. It follows the table driven parser pseudo-code given in the lecture slides. A stack is maintained and initialized to contain “START” and “$”. The next token from the lexter is then continuously compared to the top of this stack. If the next expected value is a terminal and matches the nex token, the top value of the stack is simply popped. Otherwise the grammar table is used to find a rule that matches the top value and the foundToken. The right hand side of this rule is reversed and added to the stack. This continues until the stack is empty. If at this point, the next found token from the Lexer is the end of file character, then the parse was successful.

Section 4 - Use Of Tools

The only external tools used are those mentioned in Section 1. The grammartool.jar tool by Joey Paquet was used to remove Left Recursion, EBNF Optionality and EBNF Repetition. kfG Edit was used to help find ambiguities in the grammar. The website cyberzhg.github.io was then used to help fix some of these ambiguities though I found this tool to be extremely unreliable.