Lucas Turpin (40029907) Assignment 2

COMP 442 - Presented to Prof. Joey Paquet on February 19, 2020

Section 1: Grammar

The following grammar was used as a basis to implement the parser:

```
<START> ::= <prog>

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

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

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

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

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

<assignOp> ::= '='

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

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

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

<fParams> ::= <type> 'id' <rept-fParams2> <rept-fParams3>
<fParams> ::= EPSILON

<fParamsTail> ::= ',' <type> 'id' <rept-fParamsTail3>

<factor> ::= <variable>
<factor> ::= <functionCall>
<factor> ::= 'intNum'
<factor> ::= 'floatNum'
<factor> ::= '(' <arithExpr> ')'
<factor> ::= 'not' <factor>
<factor> ::= <sign> <factor>

<funcBody> ::= <opt-funcBody0> 'do' <rept-funcBody2> 'end'
```

<funcDecl> ::= 'id' '(' <fParams> ')' ':' <type> ';'
<funcDecl> ::= 'id' '(' <fParams> ')' ':' 'void' ';'

<funcDef> ::= <funcHead> <funcBody> ';'

<funcHead> ::= <opt-funcHead0> 'id' '(' <fParams> ')' ':' <type>
<funcHead> ::= <opt-funcHead0> 'id' '(' <fParams> ')' ':' 'void'

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

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

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

<memberDecl> ::= <funcDecl>
<memberDecl> ::= <varDecl>

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

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

<opt-funcBody0> ::= 'local' <rept-opt-funcBody01>
<opt-funcBody0> ::= EPSILON

<opt-funcHead0> ::= 'id' 'sr'
<opt-funcHead0> ::= EPSILON

<prog> ::= <rept-prog0> <rept-prog1> 'main' <funcBody>

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

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

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

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

<rept-fParams2> ::= <arraySize> <rept-fParams2>
<rept-fParams2> ::= EPSILON

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

<rept-fParamsTail3> ::= <arraySize> <rept-fParamsTail3>
<rept-fParamsTail3> ::= EPSILON

<rept-funcBody2> ::= <statement> <rept-funcBody2>
<rept-funcBody2> ::= EPSILON

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

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

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

<rept-opt-funcBody01> ::= <varDecl> <rept-opt-funcBody01>
<rept-opt-funcBody01> ::= EPSILON

<rept-prog0> ::= <classDecl> <rept-prog0>
<rept-prog0> ::= EPSILON

<rept-prog1> ::= <funcDef> <rept-prog1>
<rept-prog1> ::= EPSILON

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

<rept-varDecl2> ::= <arraySize> <rept-varDecl2>
<rept-varDecl2> ::= EPSILON

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

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

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

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

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

<statBlock> ::= 'do' <rept-statBlock1> 'end'

```
<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'

<varDecl> ::= <type> 'id' <rept-varDecl2> ';'

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

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

Section 2: First and Follow Sets

First sets:

```
FIRST(<statement>)= ['write', 'return', 'id', 'if', 'read', 'while']
FIRST(<rept-opt-classDecl22>)= [',', EPSILON]
FIRST(<rept-idnest1>)= ['[', EPSILON]
FIRST(<rept-functionCall0>)= ['id', EPSILON]
FIRST(<addOp>)= ['or', '+', '-']
FIRST(<rept-opt-funcBody01>)= ['id', EPSILON, 'integer', 'float']
FIRST(<visibility>)= ['public', 'private']
FIRST(<START>)= ['id', 'main', 'class']
FIRST(<funcHead>)= ['id']
FIRST(<funcDef>)= ['id']
FIRST(<aParamsTail>)= [',']
FIRST(<funcDecl>)= ['id']
FIRST(<varDecl>)= ['id', 'integer', 'float']
FIRST(<fParamsTail>)= [',']
FIRST(<memberDecl>)= ['id', 'integer', 'float']
FIRST(<aParams>)= ['floatNum', '(', 'id', '+', EPSILON, '-', 'intNum', 'not']
FIRST(<classDecl>)= ['class']
FIRST(<fParams>)= ['id', EPSILON, 'integer', 'float']
```

FIRST(<relOp>)= ['leq', 'gt', 'neq', 'lt', 'eq', 'geq']
FIRST(<indice>)= ['[']
FIRST(<funcBody>)= ['local', 'do']
FIRST(<opt-funcHead0>)= ['id', EPSILON]
FIRST(<sign>)= ['+', '-']
FIRST(<statBlock>)= ['write', 'return', 'id', 'if', EPSILON, 'do', 'read', 'while']
FIRST(<relExpr>)= ['floatNum', '(', 'id', '+', '-', 'intNum', 'not']
FIRST(<variable>)= ['id']
FIRST(<factor>)= ['floatNum', '(', 'id', '+', '-', 'intNum', 'not']
FIRST(<prog>)= ['id', 'main', 'class']
FIRST(<term>)= ['floatNum', '(', 'id', '+', '-', 'intNum', 'not']
FIRST(<multOp>)= ['*', 'and', '/']
FIRST(<rightrec-term>)= ['*', 'and', EPSILON, '/']
FIRST(<rept-varDecl2>)= ['[', EPSILON]
FIRST(<opt-classDecl2>)= ['inherits', EPSILON]
FIRST(<rept-aParams1>)= [',', EPSILON]
FIRST(<expr>)= ['floatNum', '(', 'id', '+', '-', 'intNum', 'not']
FIRST(<idnest>)= ['id']
FIRST(<rept-fParamsTail3>)= ['[', EPSILON]
FIRST(<functionCall>)= ['id']
FIRST(<rept-classDecl4>)= ['public', EPSILON, 'private']
FIRST(<type>)= ['id', 'integer', 'float']
FIRST(<arithExpr>)= ['floatNum', '(', 'id', '+', '-', 'intNum', 'not']
FIRST(<rept-fParams2>)= ['[', EPSILON]
FIRST(<rightrec-arithExpr>)= ['or', '+', EPSILON, '-']
FIRST(<rept-fParams3>)= [',', EPSILON]
FIRST(<arraySize>)= ['[']
FIRST(<assignStat>)= ['id']
FIRST(<opt-funcBody0>)= ['local', EPSILON]
FIRST(<rept-funcBody2>)= ['write', 'return', 'id', 'if', EPSILON, 'read', 'while']
FIRST(<rept-statBlock1>)= ['write', 'return', 'id', 'if', EPSILON, 'read', 'while']
FIRST(<rept-variable0>)= ['id', EPSILON]
FIRST(<rept-prog0>)= [EPSILON, 'class']
FIRST(<rept-variable2>)= ['[', EPSILON]
FIRST(<rept-prog1>)= ['id', EPSILON]

Follow sets:

FOLLOW(<statement>)= ['write', 'return', 'else', 'id', ';', 'if', 'end', 'read', 'while']
FOLLOW(<rept-opt-classDecl22>)= ['{']
FOLLOW(<rept-idnest1>)= ['.']
FOLLOW(<rept-functionCall0>)= ['id']
FOLLOW(<addOp>)= ['floatNum', '(', 'id', '+', '-', 'intNum', 'not']
FOLLOW(<rept-opt-funcBody01>)= ['do']
FOLLOW(<visibility>)= ['id', 'integer', 'float']
FOLLOW(<START>)= [$]
FOLLOW(<funcHead>)= ['local', 'do']
FOLLOW(<funcDef>)= ['id', 'main']
FOLLOW(<aParamsTail>)= [')', ',']
FOLLOW(<funcDecl>)= ['public', '}', 'private']
FOLLOW(<varDecl>)= ['public', 'id', '}', 'do', 'private', 'integer', 'float']
FOLLOW(<fParamsTail>)= [')', ',']
FOLLOW(<memberDecl>)= ['public', '}', 'private']
FOLLOW(<aParams>)= [')']
FOLLOW(<classDecl>)= ['id', 'main', 'class']
FOLLOW(<fParams>)= [')']
FOLLOW(<relOp>)= ['floatNum', '(', 'id', '+', '-', 'intNum', 'not']
FOLLOW(<indice>)= ['leq', 'gt', 'neq', 'lt', 'or', '[', ';', ']', '=', ')', '*', 'and', '+', ',', '-', '.', '/', 'eq', 'geq']
FOLLOW(<funcBody>)= [$, ';']
FOLLOW(<opt-funcHead0>)= ['id']
FOLLOW(<sign>)= ['floatNum', '(', 'id', '+', '-', 'intNum', 'not']
FOLLOW(<statBlock>)= ['else', ';']
FOLLOW(<relExpr>)= [')', ';', ',']
FOLLOW(<variable>)= ['leq', 'gt', 'neq', 'lt', 'or', ';', ']', '=', ')', '*', 'and', '+', ',', '-', '/', 'eq', 'geq']
FOLLOW(<factor>)= ['leq', 'gt', 'neq', 'lt', 'or', ';', ']', ')', '*', 'and', '+', ',', '-', '/', 'eq', 'geq']
FOLLOW(<prog>)= [$]
FOLLOW(<term>)= ['leq', 'gt', 'neq', 'lt', 'or', ';', ']', ')', '+', ',', '-', 'eq', 'geq']
FOLLOW(<multOp>)= ['floatNum', '(', 'id', '+', '-', 'intNum', 'not']
FOLLOW(<rightrec-term>)= ['leq', 'gt', 'neq', 'lt', 'or', ';', ']', ')', '+', ',', '-', 'eq', 'geq']
FOLLOW(<rept-varDecl2>)= [';']
FOLLOW(<opt-classDecl2>)= ['{']
FOLLOW(<rept-aParams1>)= [')']
FOLLOW(<expr>)= [')', ';', ',']
FOLLOW(<idnest>)= ['id']
FOLLOW(<rept-fParamsTail3>)= [')', ',']
FOLLOW(<functionCall>)= ['leq', 'gt', 'neq', 'lt', 'or', ';', ']', ')', '*', 'and', '+', ',', '-', '/', 'eq', 'geq']
FOLLOW(<rept-classDecl4>)= ['}']
FOLLOW(<type>)= ['local', 'id', ';', 'do']
FOLLOW(<arithExpr>)= ['leq', 'gt', 'neq', 'lt', ')', ';', ',', ']', 'eq', 'geq']
FOLLOW(<rept-fParams2>)= [')', ',']
FOLLOW(<rightrec-arithExpr>)= ['leq', 'gt', 'neq', 'lt', ')', ';', ',', ']', 'eq', 'geq']
FOLLOW(<rept-fParams3>)= [')']
FOLLOW(<arraySize>)= [')', '[', ';', ',']
FOLLOW(<assignStat>)= [';']

```
FOLLOW(<opt-funcBody0>)= ['do']
FOLLOW(<rept-funcBody2>)= ['end']
FOLLOW(<rept-statBlock1>)= ['end']
FOLLOW(<rept-variable0>)= ['id']
FOLLOW(<rept-prog0>)= ['id', 'main']
FOLLOW(<rept-variable2>)= ['leq', 'gt', 'neq', 'lt', 'or', ';', ']', '=', ')', '*', 'and', '+', ',', '-', '/', 'eq', 'geq']
FOLLOW(<rept-prog1>)= ['main']
```

Section 3: Design

I have implemented a recursive descent predictive parser based on the templated implementation provided in the courses' slides. Each non-terminal is a private method on the Parser class (names have been *snake_case*d). I have added a simple heuristic to recover from errors at specific rules: if the parser fails to match for *statement, classDecl, varDecl, funcDecl* or *funcDef*, it will simply continue parsing the next one and report the error.

As I found the requirement to output the derivation proof to be a bit ambiguous, I provide two output files: *originalfilename.outderivation* and *originalfilename.outderivation.var*. The first contains the list of rules used to parse the source file in the order they are completed. The second contains the series of substitutions, starting from *prog*, required to reach the same token list as is found in the input source file.

I have chosen to encode the AST as an XML document to properly visualize nested structures as well as properties of each node in the tree.

For the AST, as Python does not have a concept of "assignable" references, I opted to either pass a container to a given function, e.g. *Parser._expr*, or to instantiate an ASTNode before calling a given function, e.g. *Parser._class_decl*. I found there was no need to manually manage a linked list of siblings/children. Instead simply store all children in a Python built-in list as an attribute of the parent. This reduces the complexity of my *ASTNode* class considerably.

Section 4: Use of Tools

I have used the provided *grammartool* as well as the online Context-Free Grammar Tool by UCalgary to convert the given grammar to an LL(1) as well as for generating the FIRST/FOLLOW sets.