Name: Minh Tuan To
Student ID: 40114920
COMP 442 Compiler Design

Section 1: Transformed grammar

<START> ::= <rept-START0>
<aParams> ::= <expr> <rept-aParams1>
<aParams> ::= EPSILON
<aParamsTail> ::= ',' <expr>
<addOp> ::= '+'
<addOp> ::= '-'
<addOp> ::= 'or'
<arithExpr> ::= <term> <rightrec-arithExpr>
<arraySize1> ::= 'intLit' ']'
<arraySize1> ::= ']'
<arraySize> ::= '[' <arraySize1>
<assignOp> ::= '='
<assignStat> ::= 'id' <assignStatIdnest> <assignOp> <expr>
<assignStat> ::= <functionCall>
<assignStatIdnest> ::= <idnest> <assignStatIdnest>
<assignStatIdnest> ::= EPSILON
<classDecl> ::= 'class' 'id' <opt-classDecl2> '{' <rept-classDecl8> '}' ';'
<classDeclOrFuncDef> ::= <classDecl>
<classDeclOrFuncDef> ::= <funcDef>
<expr1> ::= <relOp> <arithExpr>
<expr1> ::= EPSILON
<expr> ::= <arithExpr> <expr1>
<fParams> ::= 'id' ':' <type> <rept-fParams3> <rept-fParams4>
<fParams> ::= EPSILON
<fParamsTail> ::= ',' 'id' ':' <type> <rept-fParamsTail4>
<factor1> ::= <rept-variable2>
<factor1> ::= <functionCall>
<factor2> ::= <arithExpr> <factor3>
<factor2> ::= EPSILON
<factor3> ::= EPSILON
<factor3> ::= <relOp> <arithExpr> <rept-aParams1>
<factor3> ::= <aParamsTail> <rept-aParams1>
<factor> ::= <rept-idnest> 'id' <factor1>
<factor> ::= 'intLit'
<factor> ::= 'floatLit'
<factor> ::= '(' <arithExpr> ')'
<factor> ::= 'not' <factor>
<factor> ::= <sign> <factor>
<funcBody> ::= '{' <rept-funcBody1> '}'
<funcDef> ::= <funcHead> <funcBody>
<funcHead1> ::= 'sr' <funcHead2>
<funcHead1> ::= '(' <fParams> ')' 'arrow' <returnType>
<funcHead2> ::= 'id' '(' <fParams> ')' 'arrow' <returnType>

<funcHead2> ::= 'constructor' '(' <fParams> ')'
<funcHead> ::= 'function' 'id' <funcHead1>
<functionCall1> ::= <aParams>
<functionCall> ::= '(' <functionCall1> ')'
<idnest1> ::= <rept-idnest1> '.'
<idnest1> ::= '(' <aParams> ')' '.'
<idnest> ::= 'id' <idnest1>
<indice> ::= '[' <arithExpr> ']'
<localVarDecl1> ::= <rept-localVarDecl4> ';'
<localVarDecl1> ::= '(' <aParams> ')' ';'
<localVarDecl> ::= 'localVar' 'id' ':' <type> <localVarDecl1>
<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-classDecl5>
<opt-classDecl2> ::= 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-classDecl5> ::= ',' 'id' <rept-classDecl5>
<rept-classDecl5> ::= EPSILON
<rept-classDecl8> ::= <visibility> <memberDecl> <rept-classDecl8>
<rept-classDecl8> ::= 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-idnest0> ::= <idnest1> 'id' <rept-idnest0>
<rept-idnest0> ::= EPSILON
<rept-idnest1> ::= <indice> <rept-idnest1>

<rept-idnest1> ::= EPSILON
<rept-idnest> ::= <idnest> 'id' <rept-idnest0>
<rept-localVarDecl4> ::= <arraySize> <rept-localVarDecl4>
<rept-localVarDecl4> ::= EPSILON
<rept-memberVarDecl4> ::= <arraySize> <rept-memberVarDecl4>
<rept-memberVarDecl4> ::= EPSILON
<rept-statBlock1> ::= <statement> <rept-statBlock1>
<rept-statBlock1> ::= EPSILON
<rept-variable2> ::= <indice> <rept-variable3>
<rept-variable3> ::= <indice> <rept-variable3>
<rept-variable3> ::= 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
<statement0> ::= <idnest> <statement0>
<statement> ::= <rept-idnest> 'id' <assignStat> ';'
<statement> ::= 'if' '(' <relExpr> ')' 'then' <statBlock> 'else' <statBlock> ';'
<statement> ::= 'while' '(' <relExpr> ')' <statBlock> ';'
<statement> ::= 'read' '(' <statement0> <rept-variable2> ')' ';'
<statement> ::= 'write' '(' <expr> ')' ';'
<statement> ::= 'return' '(' <expr> ')' ';'
<term> ::= <factor> <rightrec-term>
<type> ::= 'integer'
<type> ::= 'float'
<type> ::= 'id'
<visibility> ::= 'public'
<visibility> ::= 'private'
<visibility> ::= EPSILON

Comment:

- I know that this grammar is not accurate, but it is LL(1) and I have not been able to fix it to become the correct grammar.
- I have tried my best to transform it but this it the result so far, and I hope there is still partial marks for it!

Section 2: FIRST and FOLLOW set

FIRST(<statement>)= ['write', 'return', 'id', 'if', 'read', 'while']
FIRST(<rept-idnest0>)= ['(', '[', EPSILON, '.']
FIRST(<rept-idnest1>)= ['[', EPSILON]
FIRST(<addOp>)= ['or', '+', '-']

FIRST(<factor1>)= ['(', '[']
FIRST(<factor2>)= ['(', 'id', '+', 'intLit', EPSILON, '-', 'floatLit', 'not']
FIRST(<factor3>)= ['leq', 'gt', 'neq', 'lt', EPSILON, ',', 'eq', 'geq']
FIRST(<returnType>)= ['id', 'integer', 'float', 'void']
FIRST(<expr1>)= ['leq', 'gt', 'neq', 'lt', EPSILON, 'eq', 'geq']
FIRST(<idnest1>)= ['(', '[', '.']
FIRST(<localVarDecl>)= ['localVar']
FIRST(<visibility>)= ['public', EPSILON, 'private']
FIRST(<START>)= ['function', EPSILON, 'class']
FIRST(<funcHead>)= ['function']
FIRST(<funcDef>)= ['function']
FIRST(<functionCall1>)= ['(', 'id', '+', 'intLit', EPSILON, '-', 'floatLit', 'not']
FIRST(<aParamsTail>)= [',']
FIRST(<arraySize1>)= ['intLit', ']']
FIRST(<memberFuncDecl>)= ['function', 'constructor']
FIRST(<memberVarDecl>)= ['attribute']
FIRST(<fParamsTail>)= [',']
FIRST(<memberDecl>)= ['function', 'constructor', 'attribute']
FIRST(<aParams>)= ['(', 'id', '+', 'intLit', EPSILON, '-', 'floatLit', 'not']
FIRST(<classDecl>)= ['class']
FIRST(<localVarDeclOrStmt>)= ['write', 'localVar', 'return', 'id', 'if', 'read', 'while']
FIRST(<fParams>)= ['id', EPSILON]
FIRST(<relOp>)= ['leq', 'gt', 'neq', 'lt', 'eq', 'geq']
FIRST(<indice>)= ['[']
FIRST(<funcBody>)= ['{']
FIRST(<statement0>)= ['id']
FIRST(<sign>)= ['+', '-']
FIRST(<rept-START0>)= ['function', EPSILON, 'class']
FIRST(<rept-idnest>)= ['id']
FIRST(<statBlock>)= ['{', 'write', 'return', 'id', 'if', EPSILON, 'read', 'while']
FIRST(<assignOp>)= ['=']
FIRST(<rept-localVarDecl4>)= ['[', EPSILON]
FIRST(<relExpr>)= ['(', 'id', '+', 'intLit', '-', 'floatLit', 'not']
FIRST(<factor>)= ['(', 'id', '+', 'intLit', '-', 'floatLit', 'not']
FIRST(<term>)= ['(', 'id', '+', 'intLit', '-', 'floatLit', 'not']
FIRST(<multOp>)= ['*', 'and', '/']
FIRST(<rightrec-term>)= ['*', 'and', EPSILON, '/']
FIRST(<opt-classDecl2>)= [EPSILON, 'isa']
FIRST(<localVarDecl1>)= ['(', '[', ';']
FIRST(<rept-memberVarDecl4>)= ['[', EPSILON]
FIRST(<rept-aParams1>)= [',', EPSILON]
FIRST(<expr>)= ['(', 'id', '+', 'intLit', '-', 'floatLit', 'not']
FIRST(<funcHead1>)= ['(', 'sr']
FIRST(<idnest>)= ['id']
FIRST(<funcHead2>)= ['constructor', 'id']
FIRST(<rept-fParamsTail4>)= ['[', EPSILON]
FIRST(<functionCall>)= ['(']
FIRST(<type>)= ['id', 'integer', 'float']
FIRST(<arithExpr>)= ['(', 'id', '+', 'intLit', '-', 'floatLit', 'not']

FIRST(<rept-classDecl5>)= [',', EPSILON]
FIRST(<rightrec-arithExpr>)= ['or', '+', EPSILON, '-']
FIRST(<rept-fParams3>)= ['[', EPSILON]
FIRST(<arraySize>)= ['[']
FIRST(<rept-classDecl8>)= ['function', 'constructor', 'public', EPSILON, 'private', 'attribute']
FIRST(<rept-fParams4>)= [',', EPSILON]
FIRST(<assignStat>)= ['(', 'id']
FIRST(<rept-funcBody1>)= ['write', 'localVar', 'return', 'id', 'if', EPSILON, 'read', 'while']
FIRST(<classDeclOrFuncDef>)= ['function', 'class']
FIRST(<assignStatIdnest>)= ['id', EPSILON]
FIRST(<rept-statBlock1>)= ['write', 'return', 'id', 'if', EPSILON, 'read', 'while']
FIRST(<rept-variable2>)= ['[']
FIRST(<rept-variable3>)= ['[', EPSILON]

FOLLOW(<statement>)= ['write', 'localVar', 'return', 'id', '}', 'else', 'if', ';', 'read', 'while']
FOLLOW(<rept-idnest0>)= ['id']
FOLLOW(<rept-idnest1>)= ['.']
FOLLOW(<addOp>)= ['(', 'id', '+', 'intLit', '-', 'floatLit', 'not']
FOLLOW(<factor1>)= ['leq', 'gt', 'neq', 'lt', 'or', ';', ']', ')', '*', 'and', '+', ',', '-', '/', 'eq', 'geq']
FOLLOW(<factor2>)= []
FOLLOW(<factor3>)= []
FOLLOW(<returnType>)= ['{', ';']
FOLLOW(<expr1>)= [')', ';', ',']
FOLLOW(<idnest1>)= ['id', '=']
FOLLOW(<localVarDecl>)= ['write', 'localVar', 'return', 'id', '}', 'if', 'read', 'while']
FOLLOW(<visibility>)= ['function', 'constructor', 'attribute']
FOLLOW(<START>)= [$]
FOLLOW(<funcHead>)= ['{']
FOLLOW(<funcDef>)= ['function', $, 'class']
FOLLOW(<functionCall1>)= [')']
FOLLOW(<aParamsTail>)= [')', ',']
FOLLOW(<arraySize1>)= [')', '[', ';', ',']
FOLLOW(<memberFuncDecl>)= ['function', 'constructor', 'public', '}', 'private', 'attribute']
FOLLOW(<memberVarDecl>)= ['function', 'constructor', 'public', '}', 'private', 'attribute']
FOLLOW(<fParamsTail>)= [')', ',']
FOLLOW(<memberDecl>)= ['function', 'constructor', 'public', '}', 'private', 'attribute']
FOLLOW(<aParams>)= [')']
FOLLOW(<classDecl>)= ['function', $, 'class']
FOLLOW(<localVarDeclOrStmt>)= ['write', 'localVar', 'return', 'id', '}', 'if', 'read', 'while']
FOLLOW(<fParams>)= [')']
FOLLOW(<relOp>)= ['(', 'id', '+', 'intLit', '-', 'floatLit', 'not']
FOLLOW(<indice>)= ['leq', 'gt', 'neq', 'lt', 'or', '[', ';', ']', ')', '*', 'and', '+', ',', '-', '.', '/', 'eq', 'geq']
FOLLOW(<funcBody>)= ['function', $, 'class']
FOLLOW(<statement0>)= ['[']
FOLLOW(<sign>)= ['(', 'id', '+', 'intLit', '-', 'floatLit', 'not']
FOLLOW(<rept-START0>)= [$]
FOLLOW(<rept-idnest>)= ['id']
FOLLOW(<statBlock>)= ['else', ';']
FOLLOW(<assignOp>)= ['(', 'id', '+', 'intLit', '-', 'floatLit', 'not']

FOLLOW(<rept-localVarDecl4>)= [';']
FOLLOW(<relExpr>)= [')']
FOLLOW(<factor>)= ['leq', 'gt', 'neq', 'lt', 'or', ';', ']', ')', '*', 'and', '+', ',', '-', '/', 'eq', 'geq']
FOLLOW(<term>)= ['leq', 'gt', 'neq', 'lt', 'or', ';', ']', ')', '+', ',', '-', 'eq', 'geq']
FOLLOW(<multOp>)= ['(', 'id', '+', 'intLit', '-', 'floatLit', 'not']
FOLLOW(<rightrec-term>)= ['leq', 'gt', 'neq', 'lt', 'or', ';', ']', ')', '+', ',', '-', 'eq', 'geq']
FOLLOW(<opt-classDecl2>)= ['{']
FOLLOW(<localVarDecl1>)= ['write', 'localVar', 'return', 'id', '}', 'if', 'read', 'while']
FOLLOW(<rept-memberVarDecl4>)= [';']
FOLLOW(<rept-aParams1>)= [')']
FOLLOW(<expr>)= [')', ';', ',']
FOLLOW(<funcHead1>)= ['{']
FOLLOW(<idnest>)= ['id', '=']
FOLLOW(<funcHead2>)= ['{']
FOLLOW(<rept-fParamsTail4>)= [')', ',']
FOLLOW(<functionCall>)= ['leq', 'gt', 'neq', 'lt', 'or', ';', ']', ')', '*', 'and', '+', ',', '-', '/', 'eq', 'geq']
FOLLOW(<type>)= ['{', '(', ')', ';', '[', ',']
FOLLOW(<arithExpr>)= ['leq', 'gt', 'neq', 'lt', ')', ';', ',', ']', 'eq', 'geq']
FOLLOW(<rept-classDecl5>)= ['{']
FOLLOW(<rightrec-arithExpr>)= ['leq', 'gt', 'neq', 'lt', ')', ';', ',', ']', 'eq', 'geq']
FOLLOW(<rept-fParams3>)= [')', ',']
FOLLOW(<arraySize>)= [')', '[', ';', ',']
FOLLOW(<rept-classDecl8>)= ['}']
FOLLOW(<rept-fParams4>)= [')']
FOLLOW(<assignStat>)= [';']
FOLLOW(<rept-funcBody1>)= ['}']
FOLLOW(<classDeclOrFuncDef>)= ['function', $, 'class']
FOLLOW(<assignStatIdnest>)= ['=']
FOLLOW(<rept-statBlock1>)= ['}']
FOLLOW(<rept-variable2>)= ['leq', 'gt', 'neq', 'lt', 'or', ';', ']', ')', '*', 'and', '+', ',', '-', '/', 'eq', 'geq']
FOLLOW(<rept-variable3>)= ['leq', 'gt', 'neq', 'lt', 'or', ';', ']', ')', '*', 'and', '+', ',', '-', '/', 'eq', 'geq']

Section 3: Design

- Similar to the first assignment designs, I utilize object-oriented to factorize and establish relationships between data types.
- The parsing table is hardcoded into my program using HashMap so that each table entry can be accessed as fast as possible
- Here are some descriptions of the classes:
    - GrammarToken.java: This is an abstract class that is the parent class for NonTerminal.java and Terminal.java. The purpose is to generalize the data structures' types and have them store both non-terminals and terminals.
    - Terminal.java: This is a class that represents terminal tokens from the grammar. In order to minimize the space complexity, I have created a static set of terminals that can be called in other classes.
    - NonTerminal.java: This is a class that represents non-terminal tokens. Each non-terminal token has variable tableEntry that can store the productions that it has. tableEntry is a HashMap with string as key and Stack as value. Since the table has columns as terminals

and rows as non-terminals, these tableEntry acts as a row, with keys (terminals) act as column id to get the corresponding table entry. The value on this map is a stack, because for the table implementation, we have to implement inverse right-hand-side push from the table entry to the grammar stack. Therefore, stack would be the best implementation for this purpose.

- ParsingTable.java: This class acts as a parsing table that is hard-coded. The underlying data structure is a hash map that stores each non-terminal with string keys. The keys are non-terminal names, and the values are the non-terminals. Since each non-terminal acts as a row, this table put all rows together to make a full-on table.
- ProgramQueue.java: This is a class that stores the output of the lexical analyzer. For each token that the lexical analyzer creates, that token will then be stored in the program queue for the syntax analyzer to parse. The underlying data structure is a queue, so the order of the program is preserved.
- GrammarStack.java: This is a class that stores non-terminals and terminals as the syntax analyzer parses the program. The underlying data structure is a stack, so the logic of popping the stack is similar to what I see in class.
- SyntaxAnalyzer.java: This is the main class of the syntax analyzer module that is responsible for checking the syntax of the program. The analyze() function is implemented with table parsing algorithm that was discussed in class. The skipError() function is also implemented with the skip error algorithm that was discussed in class.
- OutputWriter.java: This class shows up from my assignment 1, and I am reusing this class to write to output file as well. Syntax analyzer output writer has a different stream from the lexical analyzer output writer to ensure the performance and directed to different files.

Section 4: Use of tools

- Libraries:
  - Java.util.Stack: For stack implementation (parsing table, and grammar stack)
  - Java.util.Queue: For queue implementation (program queue)
  - Java.util.ArrayList : For initializing queue, and controlling line count of program queue
  - Java.util.HashMap: For parsing table implementation
  - Java.io.BufferedWriter: For the fastest writing performance for file
- Tools:
  - AtoCC: For checking LL(1) condition of the grammar.
  - UCalgary tool: For generating the parsing table.
  - grammartool.jar: For preprocessing the grammar.