Assignment 3 AST Generation

Section 1 Attribute grammar and semantic actions

<START> ::= <prog>

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

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

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

<addOp> ::= '+' sa1
<addOp> ::= '-' sa1
<addOp> ::= 'or' sa1

<arithExpr> ::= sa2 <term> <rightrec-arithExpr>

<rightrec-arithExpr> ::= <addOp> <term> <rightrec-arithExpr>
<rightrec-arithExpr> ::= EPSILON sa8

<arraySize> ::= '[' <arraySize1>
<arraySize1> ::= 'intLit' sa1 ']'
<arraySize1> ::= sa36 ']'

<assignOp> ::= '='

<expr> ::= <arithExpr> <expr1>
<expr1> ::= EPSILON sa6
<expr1> ::= <relOp> <arithExpr> sa7

<fParams> ::= sa2 'id' sa1 ':' <type> sa2 <rept-fParams3> sa3 sa27 <rept-fParams4> sa28
<fParams> ::= sa2 EPSILON sa28

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

<fParamsTail> ::= ',' 'id' sa1 ':' <type> sa2 <rept-fParamsTail4> sa3 sa27

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

<factor> ::= <funcOrVar> sa14
<factor> ::= 'intLit' sa1
<factor> ::= 'floatLit' sa1
<factor> ::= '(' <arithExpr> sa14 ')'
<factor> ::= sa2 'not' sa1 <factor> sa15
<factor> ::= sa2 <sign> <factor> sa15

<funcOrVar> ::= 'id' sa1 <funcOrVarIdnest>
<funcOrVarIdnest> ::= sa2 <rept-idnest1> sa4 sa11 <funcOrVarIdnestTail>
<funcOrVarIdnest> ::= '(' sa2 <aParams> sa5 ')' s13 <funcOrVarIdnestTail>
<funcOrVarIdnestTail> ::= '.' 'id' sa1 <funcOrVarIdnest> sa12
<funcOrVarIdnestTail> ::= EPSILON

<funcDecl> ::= <funcHead> ';'

<funcDef> ::= <funcHead> <funcBody> sa31

<funcBody> ::= '{' sa2 <rept-funcBody1> sa30 '}'

<funcHead> ::= 'func' 'id' sa1 '(' <fParams> ')' 'arrow' <returnType> sa29

<implDef> ::= 'impl' 'id' sa1 '{' sa2 <rept-implDef3> sa34 sa35 '}'

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

<multOp> ::= '*' sa1
<multOp> ::= '/' sa1
<multOp> ::= 'and' sa1

<opt-structDecl2> ::= 'inherits' 'id' sa1 <rept-opt-structDecl22>
<opt-structDecl2> ::= EPSILON
<rept-opt-structDecl22> ::= ',' 'id' sa1 <rept-opt-structDecl22>
<rept-opt-structDecl22> ::= EPSILON

<prog> ::= sa2 <rept-prog0> sa24

<relExpr> ::= <arithExpr> <relOp> <arithExpr> sa9

<relOp> ::= 'eq' sa1
<relOp> ::= 'neq' sa1
<relOp> ::= 'lt' sa1

<relOp> ::= 'gt' sa1
<relOp> ::= 'leq' sa1
<relOp> ::= 'geq' sa1

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

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

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

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

<rept-implDef3> ::= <funcDef> <rept-implDef3>
<rept-implDef3> ::= EPSILON

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

<rept-structDecl4> ::= <visibility> <memberDecl> <rept-structDecl4>
<rept-structDecl4> ::= EPSILON

<rept-varDecl4> ::= <arraySize> <rept-varDecl4>
<rept-varDecl4> ::= EPSILON

<returnType> ::= <type>
<returnType> ::= 'void' sa1

<sign> ::= '+' sa1
<sign> ::= '-' sa1

<statBlock> ::= '{' sa2 <rept-statBlock1> sa23 '}'
<statBlock> ::= sa2 <statement> sa23
<statBlock> ::= sa2 EPSILON sa23

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

<statement> ::= 'id' sa1 <s1>
<s1> ::= sa2 <rept-idnest1> sa4 sa11 <s2>
<s1> ::= '(' sa2 <aParams> sa5 ')' sa13 <s3>
<s2> ::= '.' 'id' sa1 <s1> sa12
<s2> ::= <assignOp> <expr> sa22 sa17 ';'
<s3> ::= '.' 'id' sa1 <s1> sa12
<s3> ::= sa17 ';'
<statement> ::= 'if' '(' <relExpr> ')' 'then' <statBlock> 'else' <statBlock> sa21 sa17 ';'
<statement> ::= 'while' '(' <relExpr> ')' <statBlock> sa20 sa17 ';'
<statement> ::= 'read' '(' <variable> sa16 sa17 ')' ';'
<statement> ::= 'write' '(' <expr> sa18 sa17 ')' ';'
<statement> ::= 'return' '(' <expr> sa19 sa17 ')' ';'

<structDecl> ::= 'struct' 'id' sa1 sa2 <opt-structDecl2> sa25 '{' sa2 <rept-structDecl4> sa32 '}' sa33 ';'

<structOrImplOrFunc> ::= <structDecl>
<structOrImplOrFunc> ::= <implDef>
<structOrImplOrFunc> ::= <funcDef>

<term> ::= sa2 <factor> <rightrec-term>
<rightrec-term> ::= EPSILON sa10
<rightrec-term> ::= <multOp> <factor> <rightrec-term>

<type> ::= 'integer' sa1
<type> ::= 'float' sa1
<type> ::= 'id' sa1

<variable> ::= 'id' sa1 <variableIdnest>
<variableIdnest> ::= sa2 <rept-idnest1> sa4 sa11 <variableIdnestTail>
<variableIdnest> ::= '(' sa2 <aParams> sa5 ')' sa13 '.' 'id' sa1 <variableIdnest>
<variableIdnestTail> ::= '.' 'id' sa1 <variableIdnest> sa12
<variableIdnestTail> ::= EPSILON

<varDecl> ::= 'let' 'id' sa1 ':' <type> sa2 <rept-varDecl4> sa3 sa26 ';'

<varDeclOrStat> ::= <varDecl>
<varDeclOrStat> ::= <statement>

<visibility> ::= 'public' sa1
<visibility> ::= 'private' sa1

Semantic action

sa1:makeNode(token)
sa2:pushNull
sa3:makeFamilyUntil(arraySize, popuntil)
sa4:makeFamilyUntil(rept-idnest1, popuntil)
sa5:makeFamilyUntil(aParams, popuntil)
sa6:makeFamily(expr, pop)
sa7:makeFamily(expr, pop, pop, pop)
sa8:makeFamilyUntil(arithExpr, popuntil)
sa9:makeFamily(relExpr, pop, pop, pop)
sa10:makeFamilyUntil(term, popuntil)
sa11:makeFamily(dataMem, pop, pop)
sa12:makeFamily(dot, pop, pop)
sa13:makeFamily(fCall, pop, pop)
sa14:makeFamily(factor, pop)
sa15:makeFamilyUntil(factor, popuntil)
sa16:makeFamily(readStat, pop)
sa17:makeFamily(stat, pop)
sa18:makeFamily(writeStat, pop)
sa19:makeFamily(returnStat, pop)
sa20:makeFamily(whileStat, pop, pop)
sa21:makeFamily(ifStat, pop, pop, pop)
sa22:makeFamily(assignStat, pop, pop)
sa23:makeFamilyUntil(statBlock, popuntil)
sa24:makeFamilyUntil(prog, popuntil)
sa25:makeFamilyUntil(inherlist, popuntil)
sa26:makeFamily(varDecl, pop, pop, pop)
sa27:makeFamily(fparam, pop, pop, pop)
sa28:makeFamilyUntil(fparamList, popuntil)
sa29:makeFamily(funcHead, pop, pop, pop)
sa30:makeFamily(funcBody, popuntil)
sa31:makeFamily(funcDef, pop, pop)
sa32:makeFamilyUntil(rept-structDecl4, popuntil)
sa33:makeFamily(structDecl, pop, pop, pop)
sa34:makeFamilyUntil(rept-implDef3, popuntil)
sa35:makeFamily(implDef, pop, pop)
sa36:makeNodeEmptySizeArray(token)

- makeNode: make a node with provided lexical token and push to semantic stack
- pushNull: push null to semantic stack
- makeFamilyUntil: make a node with given name and appending child by popping from the semantic stack until null is reached. Then push the new node into the stack
- makeNodeEmptySizeArray: make a node that represents arraySize, also put epsilon into the children to represent this arraySize is empty, .e.g. Int [ ]

Section 2 Design

AST generation is implemented based on Assignment 2. Node.java defines a class structure for the AST node. In Node class, it has attributes to link to the parent node and also a list to hold the child nodes. Helper function for AST also implemented in Node.java, helper function *printTree* prints the text representation of the tree structure in the console. Helper function *printTreeToFile* prints the text representation of the tree structure to the .ast.outstat file. *createDotFile* generate the graphiz dot representation of AST in .dot.outsat file. *postProcessing* removes the non-meaningful intermediate nodes from the AST.

In Parser.java, a new stack variable is added for semantic stack. The parse() function uses the attribute grammar, when a semantic action is encountered, semanticActions() is called to run for each semantic action. When the parse is done, AST is generated and the root node is stored in the semantic stack. The last step is to call Node.postProcessing to remove the unnecessary intermediate nodes such as *expr, stat, arithExpr, term, factor,* etc. Those intermedia nodes are used in attribute grammar to simplify the semantic action but do not contain semantic significance, therefore they can be removed from AST.

Section 3 Use of Tool

- GraphivizOnline: render dot file

Assignment Submission

/src - contains the source code
/input - contains the test files
/output - contains the output files
/ASTdriver.jar - executable

Jar executable is built by Maven using Java11, which requires Java Runtime Environment 11+
If you wish to rebuild the jar, use the pom.xml. Otherwise, you can compile and run the program from source code.

How to run jar executable

Run with default .src files in /input folder

```
$ java -jar ASTdriver.jar ./input/bubblesort.src
```

output files will be available in ./output