Comp 442: Compiler Design
Concordia University, Montreal
Winter 2024
William Zijie Zhang
Concordia ID: 40176870

Assignment 3

William Zijie Zhang

1 Attribute grammar

This section will indicate where each of the semantic actions are inserted in the original grammar.

```
<START> ::= <prog> 'eof'
<addOp> ::= '+' createLeaf("+")
<addOp> ::= '-' createLeaf("-")
<addOp> ::= 'or' createLeaf(or)
<aParams> ::= <expr> <rept-aParams1>
<aParams> ::= ϵ
<aParamsTail> ::= ',' <expr>
<arithExpr> ::= <term> push(ϵ) <rightrec-arithExpr>
createSubtree(rightrecArith, -1) createSubtree(arithExpr, 2)
<arraySize> ::= '[' <arraySize2>
<arraySize2> ::= 'intlit' ']' createLeaf(arraysize)
<arraySize2> ::= ']' createLeaf(emptysize)
<assignOp> ::= '=' createLeaf("=")
<expr> ::= push(ϵ) <arithExpr> <expr2> createSubtree(expr, -1)
<expr2> ::= <relOp> <arithExpr>
<expr2> ::= ϵ
<factor> ::= 'id' createLeaf(id) <factor2>
push(ϵ) <rept-var-or-funcCall>
push(createSubtree(indicelist), popuntilϵ)
<factor> ::= 'intlit' createLeaf(intLit)
<factor> ::= 'floatlit' createLeaf(floatLit)
<factor> ::= push(ϵ) '(' <arithExpr> ')'
createSubtree(arithExpr,popuntilϵ)
<factor> ::= 'not' createLeaf(not) <factor>
<factor> ::= <sign> createSubtree(sign, pop) <factor>
<factor2> ::= push(ϵ) '(' <aParams> ')'
createSubtree(paramList,popuntilϵ) createSubtree(var,pop,pop)
<factor2> ::= push(ϵ) <rept-idnest1>
createSubtree(indicelist,popuntilϵ) createSubtree(var,pop,pop)
<fParams> ::= 'id' ':' <type> createSubtree(type, pop) push(ϵ)
<rept-fParams3> createSubtree(arraySizeList,popuntilϵ)
push(ϵ) <rept-fParams4> createSubtree(paramTailList,popuntilϵ)
<fParams> ::= ϵ
```

```
<fParamsTail> ::= ',' 'id' ':' <type> createSubtree(type, pop)
push(ϵ) <rept-fParamsTail4>
createSubtree(arraySizeList,popuntilϵ)
<funcBody> ::= '{' push(ϵ) <rept-funcBody1>
createSubtree(funcBody, -1) '}'
<funcDecl> ::= <funcHead> ';'
<funcDef> ::= <funcHead> <funcBody> createSubtree(function, 4)
<funcHead> ::= 'func' 'id' createLeaf(id) push(ϵ)
'(' <fParams> ')' createSubtree(funcParams, -1)
'arrow' <returnType> createSubtree(returnType, pop)
<statement> ::= 'id' createLeaf("id") <statement-Id-nest> ';'
<statement> ::= 'if' '(' <relExpr> ')' 'then'
<statBlock> 'else' <statBlock> ';'
<statement> ::= 'while' '(' <relExpr> ')' <statBlock> ';'
<statement> ::= 'read' '(' <variable> ')' ';'
<statement> ::= 'write' '(' <expr> ')' ';'
<statement> ::= 'return' '(' <expr> ')' ';'
<statement-Id-nest> ::= '.' 'id' <statement-Id-nest>
createSubtree(dot, 2)
<statement-Id-nest> ::= '(' <aParams> ')' <statement-Id-nest2>
<statement-Id-nest> ::= <indice> <rept-idnest1>
<statement-Id-nest3>
<statement-Id-nest> ::= <assignOp> <expr>
createSubtree(assignStat, 3)
<statement-Id-nest2> ::= ϵ
<statement-Id-nest2> ::= '.' 'id' <statement-Id-nest>
createSubtree(dot, 2)
<statement-Id-nest3> ::= <assignOp> <expr>
createSubtree(assignStat, 3)
<statement-Id-nest3> ::= '.' 'id' <statement-Id-nest>
createSubtree(dot, 2)
<rept-idnest1> ::= <indice>
createSubtree(indice, pop, pop) <rept-idnest1>
<rept-idnest1> ::= ϵ
<rept-var-or-funcCall> ::= <idNest> <rept-var-or-funcCall>
<rept-var-or-funcCall> ::= ϵ
<idNest> ::= '.' 'id' createLeaf(id) <idNest2>
createSubtree(dot,pop,pop)
<idNest2> ::= push(ϵ) '(' <aParams> ')'
createSubtree(paramList,popuntilϵ) createSubtree(var,pop,pop)
<idNest2> ::= push(ϵ) <rept-idnest1>
createSubtree(indicelist,popuntilϵ)
<implDef> ::= 'impl' 'id' '{' <rept-implDef3> '}'
<indice> ::= '[' <arithExpr> ']'
<memberDecl> ::= <funcDecl> createSubtree(funcDecl, pop)
<memberDecl> ::= <varDecl> createSubtree(varDecl, pop, pop)
<multOp> ::= '*' createLeaf("*")
<multOp> ::= '/' createLeaf("/")
```

```
<multOp> ::= 'and' createLeaf(and)
<opt-structDecl2> ::= 'inherits' 'id' <rept-opt-structDecl22>
<opt-structDecl2> ::= ϵ
<prog> ::= <rept-prog0>
<relExpr> ::= <arithExpr> <relOp> <arithExpr>
createSubtree(relExpr, 3)
<relOp> ::= 'eq' createLeaf("==")
<relOp> ::= 'neq' createLeaf("!=")
<relOp> ::= 'lt' createLeaf("<")
<relOp> ::= 'gt' createLeaf(">")
<relOp> ::= 'leq' createLeaf("<=")
<relOp> ::= 'geq' createLeaf(">=")
<rept-aParams1> ::= createSubtree(factor)
<aParamsTail> <rept-aParams1>
<rept-aParams1> ::= createSubtree(factor) ϵ
<rept-fParams3> ::= <arraySize> <rept-fParams3>
<rept-fParams3> ::= ϵ
<rept-fParams4> ::= <fParamsTail> <rept-fParams4>
<rept-fParams4> ::= ϵ
<rept-fParamsTail4> ::= <arraySize> <rept-fParamsTail4>
<rept-fParamsTail4> ::= ϵ
<rept-funcBody1> ::= <varDeclOrStat> <rept-funcBody1>
<rept-funcBody1> ::= ϵ
<rept-implDef3> ::= <funcDef> <rept-implDef3>
<rept-implDef3> ::= ϵ
<rept-opt-structDecl22> ::= createLeaf(id) ',' 'id'
<rept-opt-structDecl22>
<rept-opt-structDecl22> ::= createLeaf(id) ϵ
<rept-prog0> ::= <structOrImplOrfunc> <rept-prog0>
<rept-prog0> ::= ϵ
<rept-statBlock1> ::= <statement>
createSubtree(statement, pop)<rept-statBlock1>
<rept-statBlock1> ::= ϵ
<rept-structDecl4> ::= <visibility>
createSubtree(visibility, pop) <memberDecl> <rept-structDecl4>
<rept-structDecl4> ::= ϵ
<rept-varDecl4> ::= <arraySize> <rept-varDecl4>
<rept-varDecl4> ::= ϵ
<returnType> ::= <type> createSubtree(type, pop)
<returnType> ::= 'void' createLeaf(void)
<rightrec-arithExpr> ::= ϵ
<rightrec-arithExpr> ::= <addOp> <term> <rightrec-arithExpr>
<rightRecTerm> ::= ϵ
<rightRecTerm> ::= <multOp> push(ϵ) <factor>
createSubtree(factor,popuntilϵ) <rightRecTerm>
<sign> ::= '+' createLeaf(plus)
<sign> ::= '-' createLeaf(minus)
<statBlock> ::= push(ϵ) '{' <rept-statBlock1> '}'
```

```
createSubtree(statBlock,popuntilϵ)
<statBlock> ::= <statement> push(createSubtree(statBlock, pop)
<statBlock> ::= ϵ
<structDecl> ::= 'struct' 'id' <opt-structDecl2>
'{' <rept-structDecl4> '}' ';'
<structOrImplOrfunc> ::= <structDecl> createSubtree(struct, pop)
<structOrImplOrfunc> ::= <implDef> createSubtree(impl, pop)
<structOrImplOrfunc> ::= <funcDef>
<term> ::= push(ϵ) <factor>
createSubtree(factor,popuntilϵ) push(ϵ) <rightRecTerm>
createSubtree(rightRecTerm,popuntilϵ) createSubtree(term, 2)
<type> ::= 'integer' createLeaf(integer)
<type> ::= 'float' createLeaf(float)
<type> ::= 'id' createLeaf(id)
<varDecl> ::= 'let' 'id' ':' <type> createSubtree(type, pop)
push(ϵ) <rept-varDecl4>
createSubtree(arraySizeList,popuntilϵ) ';'
<varDeclOrStat> ::= <varDecl> createSubtree(varDecl, pop, pop)
<varDeclOrStat> ::= <statement> createSubtree(statement, pop)
<variable> ::= 'id' createLeaf(id) <variable2>
<variable2> ::= push(ϵ) <rept-idnest1>
createSubtree(indicelist,popuntilϵ) createSubtree(var,pop,pop)
<rept-variable>
<variable2> ::= push(ϵ) '(' <aParams> ')'
createSubtree(paramList,popuntilϵ)
createSubtree(var,pop,pop) <var-idNest>
<rept-variable> ::= <var-idNest> <rept-variable>
<rept-variable> ::= ϵ
<var-idNest> ::= '.' 'id' createLeaf(id) <var-idNest2>
createSubtree(dot,pop,pop)
<var-idNest2> ::= push(ϵ) '(' <aParams> ')'
createSubtree(paramList,popuntilϵ)
createSubtree(var,pop,pop) <var-idNest>
<var-idNest2> ::= push(ϵ) <rept-idnest1>
createSubtree(indicelist,popuntilϵ)
<visibility> ::= 'public' createLeaf(public)
<visibility> ::= 'private' createLeaf(private)
```

2 Design

The design for the third assignment, once again, involved a lot of preparation before diving into the code. I have already demonstrated my modified attribute grammar in the previous section, so I will now focus on the implementation. To separate the assignment progress, I decided to create a copy of my original parser before making any significant changes. Then I added a class attribute for the semantic stack, which would store all the semantic concepts of our AST tree. From our attribute grammar, we only ever used two functions: *createLeaf* and *createSubtree*, I implemented those by passing a string and

a list of children to the *Node* constructor. Finally, I reused some example code from the anytree library to output the root tree in a readable format along with an image generated from graphviz.

3 Tools

I have used the Python library anytree to help me in the generation of the AST trees. I chose this library because, in addition to an easy-to-use node creation API, anytree allows me to automatically convert my trees to the *.dot* format. This enables me to automatically generate graph representations of the AST tree formed from my semantic stack. Finally, as usual, I have used examples and guidance from the lectures by Dr. Joey Paquet.