Maira Malhi (40128269)
Assignment 3 - COMP 442

**Attribute Grammar and Relevant Cases for Semantic Actions**

```
case "SACT1" -> this.makeNode(previousToken);
case "SACT2" -> this.makeNull();
case "SACT3" -> this.makeFamily("array Size");
case "SACT4" -> this.makeFamily("local var");
case "SACT5" -> this.makeFamily("class decl");
case "SACT6" -> this.makeFamily("memberVar decl");
case "SACT7" -> this.makeFamily("memberFunc decl");
case "SACT8" -> this.makeFamily("func def");
case "SACT9" -> this.makeFamily("inherit lst");
case "SACT10" -> this.makeFamily("func params");
case "SACT11" -> this.makeFamily("local var + stat block");
case "SACT12"  -> this.makeFamily("scope res");
case "SACT13"  -> this.makeFamily("stat");
case "SACT14"  -> this.makeFamily("if block");
case "SACT15"  -> this.makeFamily("while block");
case "SACT16"  -> this.makeFamily("read block");
case "SACT17"  -> this.makeFamily("write block");
case "SACT18"  -> this.makeFamily("return stat");
case "SACT19"  -> this.makeFamily("then block");
case "SACT20"  -> this.makeFamily("else stat");
```

```
ADDOP -> plus  .
ADDOP -> minus  .
ADDOP -> or  .

APARAMS -> EXPR REPTAPARAMS1  .
APARAMS ->  .

APARAMSTAIL -> comma EXPR  .

ARITHEXPR -> TERM RIGHTRECARITHEXPR  .

ARRAYOROBJECT -> SACT2 REPTARRAYSIZE SACT3 .
ARRAYOROBJECT -> lpar APARAMS rpar  .

ARRAYSIZE -> lsqbr ARRAYSIZE2  .

ARRAYSIZE2 -> intlit SACT1 rsqbr  .
ARRAYSIZE2 -> rsqbr .

ASSIGNOP -> equal  .
```

```
CLASSDECL -> class id SACT1 SACT2 OPTINHERITS SACT9 lcurbr REPTMEMBERDECL
rcurbr semi  .

CLASSDECLORFUNCDEF -> SACT2 CLASSDECL SACT5 .
CLASSDECLORFUNCDEF -> SACT2 FUNCDEF .

EXPR -> ARITHEXPR EXPR2  .

EXPR2 -> RELOP ARITHEXPR  .
EXPR2 ->  .

FACTOR -> id FACTOR2 REPTVARIABLEORFUNCTIONCALL  .
FACTOR -> intlit  .
FACTOR -> floatlit  .
FACTOR -> lpar ARITHEXPR rpar  .
FACTOR -> not FACTOR  .
FACTOR -> SIGN FACTOR  .

FACTOR2 -> lpar APARAMS rpar  .
FACTOR2 -> REPTIDNEST1  .

FPARAMS -> SACT2 id SACT1 colon TYPE REPTFPARAMS3 REPTFPARAMS4 SACT10 .
FPARAMS -> .

FPARAMSTAIL -> comma id colon TYPE REPTFPARAMSTAIL4  .

FUNCBODY -> lcurbr SACT2 REPTLOCALVARORSTAT SACT11 rcurbr  .

FUNCDEF -> FUNCHEAD SACT8 FUNCBODY  .

FUNCHEAD -> function id SACT1 FUNCHEADTAIL  .

FUNCHEADMEMBERTAIL -> id SACT1 lpar FPARAMS rpar arrow RETURNTYPE  .
FUNCHEADMEMBERTAIL -> constructorkeyword lpar FPARAMS rpar  .

FUNCHEADTAIL -> sr FUNCHEADMEMBERTAIL  .
FUNCHEADTAIL -> lpar FPARAMS rpar arrow RETURNTYPE  .

IDNEST -> dot id IDNEST2  .

IDNEST2 -> lpar APARAMS rpar  .
IDNEST2 -> REPTIDNEST1  .

INDICE -> lsqbr ARITHEXPR rsqbr  .

LOCALVARDECL -> SACT2 localvar id SACT1 colon TYPE ARRAYOROBJECT semi SACT4 .

LOCALVARORSTAT -> LOCALVARDECL  .
LOCALVARORSTAT -> SACT2 STATEMENT SACT13  .
```

```
MEMBERDECL -> SACT2 SACT1 MEMBERFUNCDECL SACT7 .
MEMBERDECL -> SACT2 SACT1 MEMBERVARDECL SACT6 .

MEMBERFUNCDECL -> MEMBERFUNCHEAD semi  .

MEMBERFUNCHEAD -> function id SACT1 colon lpar FPARAMS rpar arrow RETURNTYPE  .
MEMBERFUNCHEAD -> constructorkeyword colon lpar FPARAMS rpar  .

MEMBERVARDECL -> attribute id SACT1 colon TYPE SACT2 REPTARRAYSIZE SACT3 semi
.

MULTOP -> mult  .
MULTOP -> div  .
MULTOP -> and  .

OPTINHERITS -> isa id SACT1 REPTINHERITSLIST  .
OPTINHERITS ->  .

PROG -> REPTPROG0  .

RELEXPR -> ARITHEXPR RELOP ARITHEXPR  .

RELOP -> eq  .
RELOP -> neq  .
RELOP -> lt  .
RELOP -> gt  .
RELOP -> leq  .
RELOP -> geq  .

REPTAPARAMS1 -> APARAMSTAIL REPTAPARAMS1  .
REPTAPARAMS1 ->  .

REPTARRAYSIZE -> ARRAYSIZE REPTARRAYSIZE  .
REPTARRAYSIZE ->  .

REPTFPARAMS3 -> ARRAYSIZE REPTFPARAMS3  .
REPTFPARAMS3 ->  .

REPTFPARAMS4 -> FPARAMSTAIL REPTFPARAMS4  .
REPTFPARAMS4 ->  .

REPTFPARAMSTAIL4 -> ARRAYSIZE REPTFPARAMSTAIL4  .
REPTFPARAMSTAIL4 ->  .

REPTIDNEST1 -> INDICE REPTIDNEST1  .
REPTIDNEST1 ->  .

REPTINHERITSLIST -> comma id REPTINHERITSLIST  .
REPTINHERITSLIST ->  .

REPTLOCALVARORSTAT -> LOCALVARORSTAT REPTLOCALVARORSTAT  .
REPTLOCALVARORSTAT ->  .

REPTMEMBERDECL -> VISIBILITY MEMBERDECL REPTMEMBERDECL  .
REPTMEMBERDECL ->  .

REPTPROG0 -> CLASSDECLORFUNCDEF REPTPROG0  .
REPTPROG0 ->  .

REPTSTATBLOCK1 -> STATEMENT REPTSTATBLOCK1  .
REPTSTATBLOCK1 ->  .

REPTVARIABLE -> VARIDNEST REPTVARIABLE  .
REPTVARIABLE ->  .

REPTVARIABLEORFUNCTIONCALL -> IDNEST REPTVARIABLEORFUNCTIONCALL  .
REPTVARIABLEORFUNCTIONCALL ->  .

RETURNTYPE -> TYPE  .
RETURNTYPE -> void SACT1 .

RIGHTRECARITHEXPR -> ADDOP TERM RIGHTRECARITHEXPR  .
RIGHTRECARITHEXPR ->  .

RIGHTRECTERM -> MULTOP FACTOR RIGHTRECTERM  .
RIGHTRECTERM ->  .

SIGN -> plus  .
SIGN -> minus  .

START -> PROG eof  .

STATBLOCK -> lcurbr REPTSTATBLOCK1 rcurbr  .
STATBLOCK -> STATEMENT  .
STATBLOCK ->  .

STATEMENT -> id SACT1 STATEMENTIDNEST semi  .
STATEMENT -> SACT2 if lpar RELEXPR rpar then STATBLOCK else STATBLOCK semi
SACT14 .
STATEMENT -> SACT2 while lpar RELEXPR rpar STATBLOCK semi SACT15 .
STATEMENT -> SACT2 read lpar VARIABLE rpar semi SACT16 .
STATEMENT -> SACT2 write  lpar EXPR rpar semi SACT17 .
STATEMENT -> SACT2 return lpar EXPR rpar semi SACT18 .

STATEMENTIDNEST -> dot id STATEMENTIDNEST  .
STATEMENTIDNEST -> lpar APARAMS rpar STATEMENTIDNEST2  .
STATEMENTIDNEST -> INDICE REPTIDNEST1 STATEMENTIDNEST3  .
STATEMENTIDNEST -> ASSIGNOP EXPR  .

STATEMENTIDNEST2 ->  .
STATEMENTIDNEST2 -> dot id STATEMENTIDNEST  .

STATEMENTIDNEST3 -> ASSIGNOP EXPR  .
STATEMENTIDNEST3 -> dot id STATEMENTIDNEST  .

TERM -> FACTOR RIGHTRECTERM  .

TYPE -> integer SACT1 .
TYPE -> float SACT1 .
TYPE -> id SACT1 .

VARIABLE -> id VARIABLE2  .

VARIABLE2 -> REPTIDNEST1 REPTVARIABLE  .
VARIABLE2 -> lpar APARAMS rpar VARIDNEST  .

VARIDNEST -> dot id VARIDNEST2  .

VARIDNEST2 -> lpar APARAMS rpar VARIDNEST  .
VARIDNEST2 -> REPTIDNEST1  .

VISIBILITY -> public  .
VISIBILITY -> private  .
```

**Design**

The way I went about creating the semantic stack in the Parser was I have a node class, called AST, in which a node is pushed onto the stack whenever a semantic action (SACT) is found in the non-terminal grammar that I added in manually. The actions vary depending on which type of semantic was found for the corresponding code, as seen above.

Regarding creating the nodes, I have three methods, makeNode, makeFamily and makeNull as seen in class. The makeNode method will create a new node, and it is used to process ids. The makeFamily will allow for children nodes to be created and set to a parent node and pushed onto the semantic stack. The way it is used, I pop the last semantics found until a epsilon/ null semantic (triggered from makeNull) is found, to indicate that all children of a parent node were found.

I never process the semantic actions as tokens, they simply get caught by parser, pushed to semantic stack and popped from the token stack, and then the top of stack (a non terminal) is evaluated in the parser.

I used a string builder to build and then store the formatted tree version of my AST nodes.I simply print the stack to display the tree.

## Tools

For the tools, I previously used the University of Calgary tool to get the tables into a CSV format to delimit using an online convert csv tool that was used for my Java language to work with its grammar. However, I quickly realized I could simply manually edit my CSV in IntelliJ with any changes as I carefully ensured that semantic actions aren't being treated as parsing elements. Thus, no new tools were used.