Report
Compiler Design
comp-6421

Assignment 3
Abstract Syntax Tree

Professor: Dr. Joey Paquet

Neona Sheetal Pinto
40172626

Attribute Grammar:

```
START -> PROG  .
PROG -> REPTPROG0 sa-03 .
REPTPROG0 -> STRUCTORIMPLORFUNC REPTPROG0  .
REPTPROG0 ->  .
STRUCTORIMPLORFUNC -> STRUCTDECL .
STRUCTORIMPLORFUNC -> IMPLDEF .
STRUCTORIMPLORFUNC -> FUNCDEF .

STRUCTDECL -> struct id sa-09 OPTSTRUCTDECL2 lcurbr REPTSTRUCTDECL4 sa-08
rcurbr semi sa-04  .
OPTSTRUCTDECL2 -> inherits id sa-06 REPTOPTSTRUCTDECL22 sa-07  .
OPTSTRUCTDECL2 ->  . sa-57
REPTOPTSTRUCTDECL22 -> comma id sa-06 REPTOPTSTRUCTDECL22  .
REPTOPTSTRUCTDECL22 ->  .
REPTSTRUCTDECL4 -> VISIBILITY MEMBERDECL sa-54 REPTSTRUCTDECL4  .
REPTSTRUCTDECL4 ->  .
VISIBILITY -> public sa-55 .
VISIBILITY -> private sa-55 .
MEMBERDECL -> FUNCDECL sa-10 .
MEMBERDECL -> VARDECL .

VARDECL -> let id sa-06 colon TYPE REPTVARDECL4 semi sa-11 .
TYPE -> integer sa-13 .
TYPE -> float sa-13 .
TYPE -> id sa-06 sa-53 .
REPTVARDECL4 -> ARRAYSIZE REPTVARDECL4  .
REPTVARDECL4 ->  . sa-14
ARRAYSIZE -> lsqbr ARRAYSIZE1  .
ARRAYSIZE1 -> intnum sa-15 rsqbr  .
ARRAYSIZE1 -> sa-58 rsqbr  .

FUNCDECL -> FUNCHEAD semi  .
FUNCHEAD -> func id sa-06 lpar FPARAMS sa-12 rpar arrow RETURNTYPE  .
FPARAMS -> id sa-06 colon TYPE REPTFPARAMS3 sa-16 REPTFPARAMS4  .
FPARAMS ->  .
RETURNTYPE -> TYPE  .
RETURNTYPE -> void sa-13 .
REPTFPARAMS3 -> ARRAYSIZE REPTFPARAMS3  .
REPTFPARAMS3 ->  . sa-14
REPTFPARAMS4 -> FPARAMSTAIL REPTFPARAMS4  .
REPTFPARAMS4 ->  .
FPARAMSTAIL -> comma id sa-06 colon TYPE REPTFPARAMSTAIL4 sa-16 .
REPTFPARAMSTAIL4 -> ARRAYSIZE REPTFPARAMSTAIL4  .
REPTFPARAMSTAIL4 ->  . sa-14

FUNCDEF -> FUNCHEAD FUNCBODY sa-17 .

FUNCBODY -> lcurbr REPTFUNCBODY1 sa-21 rcurbr sa-18 .
REPTFUNCBODY1 -> VARDECLORSTAT REPTFUNCBODY1  .
REPTFUNCBODY1 ->  .
VARDECLORSTAT -> VARDECL .
VARDECLORSTAT -> STATEMENT .
STATEMENT -> if lpar EXPR sa-30 rpar then STATBLOCK sa-21 else STATBLOCK sa-21
```

```
semi sa-22 .
STATEMENT -> while lpar EXPR sa-30 rpar STATBLOCK sa-21 semi sa-23 .
STATEMENT -> read lpar VARIABLE sa-31 rpar semi sa-24 .
VARIABLE -> id sa-06 VARIABLE1  .
VARIABLE1 -> VARIABLETAIL sa-52 VARIABLE2  .
VARIABLE1 -> FUNCTIONCALLTAIL sa-49 VARIABLE3  .
VARIABLE2 -> dot id sa-06 VARIABLE1 sa-61  .
VARIABLE2 ->  .
VARIABLE3 -> dot id sa-06 VARIABLE1 sa-61 .

STATEMENT -> write lpar EXPR sa-30 rpar semi sa-25 .
STATEMENT -> return lpar EXPR sa-30 rpar semi sa-26  .
STATBLOCK -> lcurbr REPTSTATBLOCK1 rcurbr .
STATBLOCK -> STATEMENT .
STATBLOCK ->  .
REPTSTATBLOCK1 -> STATEMENT REPTSTATBLOCK1 .
REPTSTATBLOCK1 ->  .

EXPR -> ARITHEXPR EXPR1 .
EXPR1 ->  .
EXPR1 -> sa-30 RELOP ARITHEXPR sa-30 sa-34 .
RELOP -> eq sa-45 .
RELOP -> neq sa-45 .
RELOP -> lt sa-45 .
RELOP -> gt sa-45 .
RELOP -> leq sa-45 .
RELOP -> geq sa-45 .

ARITHEXPR -> TERM sa-35 RIGHTRECARITHEXPR sa-33  .
RIGHTRECARITHEXPR ->  .
RIGHTRECARITHEXPR -> sa-33 ADDOP TERM sa-35 sa-37 RIGHTRECARITHEXPR  .
ADDOP -> plus sa-47 .
ADDOP -> minus sa-47 .
ADDOP -> or sa-47 .

TERM -> FACTOR sa-36 RIGHTRECTERM  .
RIGHTRECTERM ->  .
RIGHTRECTERM -> sa-35 MULTOP FACTOR sa-36 sa-38 RIGHTRECTERM  .
MULTOP -> mult sa-48 .
MULTOP -> div sa-48 .
MULTOP -> and sa-48 .

FACTOR -> intnum sa-39 .
FACTOR -> floatnum sa-66 .
FACTOR -> lpar ARITHEXPR sa-30 rpar  .
FACTOR -> not FACTOR sa-36 sa-41 .
FACTOR -> SIGN FACTOR sa-36 sa-42 .
SIGN -> plus sa-46 .
SIGN -> minus sa-46 .

FACTOR -> id sa-06 FACTOR1 sa-44 .
FACTOR1 -> VARIABLETAIL sa-52 sa-63 sa-65 FACTOR2  .
FACTOR1 -> FUNCTIONCALLTAIL sa-49 FACTOR2  .
FACTOR2 -> dot id sa-06 FACTOR1  .
FACTOR2 ->  .
VARIABLETAIL -> INDICE VARIABLETAIL  .
```

```
VARIABLETAIL ->  . sa-51
INDICE -> lsqbr ARITHEXPR sa-30 rsqbr  .
FUNCTIONCALLTAIL -> sa-62 lpar sa-29 APARAMS rpar .
APARAMS -> EXPR sa-30 REPTAPARAMS1  .
APARAMS ->  . sa-50
REPTAPARAMS1 -> APARAMSTAIL REPTAPARAMS1  .
REPTAPARAMS1 ->  . sa-50
APARAMSTAIL -> comma EXPR sa-30  .

STATEMENT -> id sa-06 STATEMENT1 semi  .
STATEMENT1 -> VARIABLETAIL sa-52 sa-63 STATEMENT2  .
STATEMENT1 -> FUNCTIONCALLTAIL sa-49 sa-64 STATEMENT3  .
STATEMENT2 -> dot id sa-06 STATEMENT1  .
STATEMENT2 -> ASSIGNOP sa-31 EXPR sa-30 sa-32 .
STATEMENT3 -> dot id sa-06 STATEMENT1  .
STATEMENT3 ->  . sa-60
ASSIGNOP -> assign .

IMPLDEF -> impl id sa-09 lcurbr REPTIMPLDEF3 sa-05 rcurbr sa-27  .
REPTIMPLDEF3 -> FUNCDEF REPTIMPLDEF3  .
REPTIMPLDEF3 ->  .

ASSIGNSTAT -> VARIABLE ASSIGNOP EXPR  .
IDNEST -> id IDNEST1 dot  .
IDNEST1 -> REPTIDNEST1  .
REPTIDNEST1 -> INDICE REPTIDNEST1  .
REPTIDNEST1 ->  .
IDNEST1 -> lpar APARAMS rpar  .
RELEXPR -> ARITHEXPR RELOP ARITHEXPR  .
```

Semantic Actions

```
"Sa-03" -> "makeFamily(Prog, StructDecl_s, FuncDef_s, ImplDef_s, n)"
"Sa-01" -> "StructList_s", "makeFamily(StructList, StructDecl_s, n)"
"Sa-02" -> "ImplDefList_s", "makeFamily(ImplDefList, ImplDef_s,n)"
"Sa-27" -> "ImplDef_s", "makeFamily(ImplDef, Id_i, FuncDefList_s, n)"
"Sa-09" -> "Id_i", "makeNode(Id)"
"Sa-06" -> "Id_s", "makeNode(Id)"
"Sa-04" -> "StructDecl_s", "makeFamily(StructDecl, Id_i, InheritList_s,
MembList_s, n)"
"Sa-07" -> "InheritList_s", "makeFamily(InheritList, Id_s, n)"
"Sa-08" -> "MembList_s", "makeFamily(MembList, MembDecl_s, n)"
"Sa-57" -> "InheritList_s", "makeNode(InheritList)"
"Sa-54" -> "MembDecl_s", "makeFamily(MembDecl, Visibility_s, VarDecl_s,
FuncDecl_s, 2, any)"
"sa-55" -> "Visibility_s", "makeNode(Visibility)"
"sa-11" -> "VarDecl_s", "makeFamily(VarDecl,Id_s, Type_s, DimList_s)"
"sa-13" -> "Type_s", "makeNode(Type)"
"Sa-53" -> "Type_s", "makeFamily(Type, Id_s)"
"Sa-14" -> "DimList_s", "makeFamily(DimList, Dim_s, n)"
"Sa-15" -> "Dim_s", "makeNode(Dim)"
"Sa-58" -> "Dim_s", "makeNode(DimNull)"
```

```
"sa-10" -> "FuncDecl_s", "makeFamily(FuncDecl, Id_s, FParamList_s, Type_s)"
"sa-12" -> "FParamList_s", "makeFamily(FParamList, FParam_s, n)"
"sa-16" -> "FParam_s", "makeFamily(FParam, Id_s, Type_s,  DimList_s)"
"sa-05" -> "FuncDefList_s", "makeFamily(FuncDefList, FuncDef_s, n)"
"sa-17" -> "FuncDef_s", "makeFamily(FuncDef, Id_s, FParamList_s, Type_s,
FuncBody_s)"
"sa-18" -> "FuncBody_s", "makeFamily(FuncBody, VarDecl_s, StatBlock_s, n)"
"Sa-30" -> "Expr_s", "makeFamily(Expr, ArithExpr_s, RelExpr_s, any)"
"sa-33" -> "ArithExpr_s", "makeFamily(ArithExpr, Term_s, AddOp_s, any)"
"Sa-35" -> "Term_s", "makeFamily(Term, Factor_s, MultOp_s, any)"
"Sa-36" -> "Factor_s", "makeFamily(Factor, Num_s, Float_s, FuncOrVar_s,
Expr_s, Not_s, Sign_s, any)"
"sa-44" -> "FuncOrVar_s", "makeFamily(FuncOrVar, DataMem_s, Dot_s,
FuncCall_s, any)"
"sa-52" -> "DataMem_s", "makeFamily(DataMem, Id_s, Indice_s)"
"sa-51" -> "Indice_s", "makeFamily(Indice, Expr_s, n)"
"Sa-63" -> "Dot_s", "makeFamily(Dot, DataMem_s, Dot_s, DataMem_s,
keepOrSkip, first2, any)"    // maybe a dot or just a dataMem special case
"sa-65" -> "Dot_s", "makeFamily(Dot, FuncCall_s, DataMem_s, keepOrSkip)"
"Sa-62" -> "Dot_s", "makeFamily(Dot, DataMem_s, Id_s, keepOrSkip)"
 // if not DataMem, skip; if DataMem, combine
 "sa-29" -> "Null_s", "makeNode(Null)"    // as a EPSILON
 "sa-49" -> "FuncCall_s", "makeFamily(FuncCall, Id_s, Dot_s, AParams_s,
first2, any)"
 "sa-64" -> "FuncCall_s", "makeFamily(FuncCall, Dot_s, DataMem_s,
FuncCall_s, keepOrSkip, first2, any)"     // if not DataMem, just skip; if
DataMem, combine
 "sa-60" -> "FuncCallStat_s", "makeFamily(FuncCallStat, FuncCall_s, n)"
 "sa-50" -> "AParams_s", "makeFamily(AParams, Null_s, Expr_s, n)"
 "sa-39" -> "Num_s", "makeNode(Num)"
 "sa-66" -> "Float_s", "makeNode(Float)"
 "sa-41" -> "Not_s", "makeFamily(Not, Factor_s)"
 "sa-42" -> "Sign_s", "makeFamily(Sign_i, Factor_s, reuse)"
 "sa-46" -> "Sign_i", "makeNode(Sign)"
 "sa-38" -> "MultOp_s", "makeFamily(MultOp_i, Term_s, MultOp_i, Factor_s,
reuse)"
 "sa-48" -> "MultOp_i", "makeNode(MultOp)"
 "sa-37" -> "AddOp_s", "makeFamily(AddOp_i, ArithExpr_s, AddOp_i, Term_s,
reuse)"
 "sa-47" -> "AddOp_i", "makeNode(AddOp)"
 "sa-34" -> "RelExpr_s", "makeFamily(RelExpr, Expr_s, RelOp_s, Expr_s)"
 "sa-45" -> "RelOp_s", "makeNode(RelOp)"
 "sa-21" -> "StatBlock_s", "makeFamily(StatBlock, VarDecl_s, IfStat_s,
WhileStat_s, ReadStat_s, WriteStat_s, ReturnStat_s, FuncCallStat_s,
AssignStat_s, n)"
 "sa-22" -> "IfStat_s", "makeFamily(IfStat, Expr_s, StatBlock_s,
StatBlock_s)"
 "sa-23" -> "WhileStat_s", "makeFamily(WhileStat, Expr_s, StatBlock_s)"
 "sa-24" -> "ReadStat_s", "makeFamily(ReadStat, Variable_s)"
 "sa-31" -> "Variable_s", "makeFamily(Variable, DataMem_s, Dot_s, any)"
 "sa-61" -> "Dot_s", "makeFamily(Dot, DataMem_s, Dot_s, DataMem_s, 2, any)"
 "sa-25" -> "WriteStat_s", "makeFamily(WriteStat, Expr_s)"
 "sa-26" -> "ReturnStat_s", "makeFamily(ReturnStat, Expr_s)"
 "sa-32" -> "AssignStat_s", "makeFamily(AssignStat, Variable_s, Expr_s)"
```

Design:

The design approach used for the syntax analyzer is the Table-driven predictive parsing as opposed to the recursive descent approach.
The parser table is generated using the online "*ucalgary tool*" as per the parser algorithm. The parsing table which is created using the FIRST and FOLLOW sets tells the parser which right-hand-side of the rule to choose when the top of the stack is non-terminal.

The *Rule class* stores the rules in the form of the left-hand-side part and right-hand-side part.
The *SemanticAction* class stores semantic actions in the form of the left-hand-side part and right-hand-side part.
The *Grammar class* separates information about the grammar with the parser. It utilizes the parsing table, FIRST, and FOLLOW sets created manually in the class that is derived from the use of tools. The files are placed in the resource folder. The "LL1.attrbute.grm" in the folder is the grammar rules injected with semantic actions.

The *SyntacticAnalyzer(parser) class* is responsible for parsing every test case, and writing to the output files (derivation output, AST output, error reporting). It contains a LexicalAnalyzer object which is implemented in Lexical Analyser from assignment 1 for tokenizing the src file. A Grammar object and Node factory is aggregated in it. Using the stacks (parsing stack and semantic stack) is utilized to implement the parsing and semantic actions of the test case.

The *NodeFactory* class is dealing with the creation of different nodes depending on the different parameters. A package of AST is created with a group of nodes that represent the abstract syntactic structure of source code. A node class is the parent class that implements several methods(makeSiblings, adoptChildren, etc).

The *SyntacticAnalyzerDriver class* is a driver class to run the syntax analyzer which either opens every file(.src) in a folder or a single file(.src) and handles the task of parsing and generating output files.
Note: A more detailed explanation is given in the source code through comments.

Use of Tools

1. ***IntelliJ IDEA***: IDE platform for writing the source code. The IDE has smooth, efficient, and clever code suggestions which makes it easy to type. Debugging is much faster compared to other IDEs.
2. ***GraphViz Online:*** Used the tool to put the dot file from the output to generate the AST. The tool is user-friendly and helps to load the tree immediately and is flexible in viewing as well.
3. ***Java.util***: Used util files in the source code for creating tables, hashmaps,. etc.