Μεταγλωττιστές

Άσκηση Μέρος 3

Συντακτική Ανάλυση, Δημιουργία Αφηρημένου Συντακτικού Δέντρου

Στο μέρος αυτό συνεχίζουμε την υλοποίηση ενός συντακτικού αναλυτή, ο οποίος υποστηρίζει την χρήση βασικών συντακτικών κανόνων που ακολουθούν τους κανόνες της C γλώσσας. Ένας LALR(1) συντακτικός αναλυτής δημιουργείται με το εργαλείο CUP. Σε αυτό το μέρος γίνεται και η παραγωγή του αφηρημένου συντακτικού δέντρου με “μετάφραση οδηγούμενη από το συντακτικό”. Κάνουμε μια πλήρη περιγραφή της γλώσσας με βάση:

- Τις λεκτικές μονάδες που έχουμε σαν δεδομένες από το προηγούμενο βήμα, και την σημασία που αυτές έχουν, σε γλώσσες που συντακτικά βασίζονται στη C ως τερματικά σύμβολα μιας γραμματικής.
- Την δεδομένη πλήρη γραμματική της εκφώνησης:
  *<CompilationUnit> ::= (<VariableDefinition> | <FunctionDefinition>)**
  <FunctionDefinition> ::= <TypeSpecifier> <Identifier> ”(” <ParameterList>? ”)” ”{”
  <Statement> ”}”*
  <ParameterList> ::= <ParameterDeclaration>
  | <ParameterList> ”,” <ParameterDeclaration>
  <ParameterDeclaration> ::= <TypeSpecifier> <Identifier>
  | <TypeSpecifier> ”[” ”]” <Identifier>
  <TypeSpecifier> ::= ”char”
  | ”int”
  | ”float”
  | ”void”
- Τις ιδιότητες που έχει κάθε σύμβολο, όπως ο ρόλος του σε εκφράσεις και η προτεραιότητα και η προσεταιριστικότητα που αναμένεται να έχουν σε γλώσσες βασισμένες στη C.

Δομή του Project

Το project παραμένει βασισμένο στη δομή του εργαστηρίου. Το αρχείο lexer.flex έχει τον κώδικα για το εργαλείο jflex. Το αρχείο grammar.cup έχει τον κώδικα για το εργαλείο CUP. Ο κώδικας στα αρχεία αυτά μεταφράζεται σε κώδικα java αυτόματα από τα εργαλεία.
Στον Java κώδικα υπάρχει και μία υλοποίηση του ASTVisitor και έχουμε δημιουργήσει το ast directory στο οποίο βρίσκονται όλες οι κλάσεις σχετικές με το αφηρημένο συντακτικό δέντρο. Ουσιαστικά είναι τα Java πακέτα ast όπου βρίσκονται όλες οι abstract κλάσεις, οι υποκλάσεις του Declaration και τα Enumerations, ast.exprs που είναι όλες οι υποκλάσεις του Expression από το ast, και το ast.statements όπου είναι όλες οι υποκλάσεις του Statement εκτός από τα assignments τα οποία είναι στο βασικό πακέτο ast.

Περιγραφή γλώσσας στον συντακτικό αναλυτή

Το αρχείο με τον κώδικα για το εργαλείο CUP στην αρχή έχει τον κώδικα για το logging λαθών που έχει ο parser που παράγεται, με τη χρήση του Simple Logging Facade για JAVA. Ο κώδικας αυτός είναι ίδιος με του εργαστηρίου.

Στη συνέχεια δηλώνουμε τερματικά σύμβολα, τα οποία είναι είτε σταθερές, είτε identifier είτε literals:

```
//Main Terminal Symbols
terminal java.lang.String IDENTIFIER;

terminal PRINT, IF, ELSE, RETURN, NEW;
terminal DO , WHILE, BREAK, CONTINUE;
terminal INT, FLOAT, CHAR, VOID;
terminal LPAREN, RPAREN, SEMICOLON, COMMA;
terminal EQ;
terminal LESS, LESS_EQ, GREATER, GREATER_EQ, EQUAL, NOT_EQUAL;
terminal MULTI, DIV, MOD;
terminal AND, OR, NOT;
terminal PLUS, MINUS, UMINUS;
terminal LBRACKET, RBRACKET, LCURBRACKET, RCURBRACKET;

//Literals
terminal java.lang.Integer INTEGER_LITERAL;
terminal java.lang.Double  DOUBLE_LITERAL;
terminal java.lang.String  STRING_LITERAL;
terminal java.lang.Char CHARACTER_LITERAL;
```

Τα μη τερματικά ομαδοποιούν και συνδυάζουν τερματικά και μη τερματικά με βάση τους κανόνες που θα δούμε στη γραμματική.Κάθε μη τερματικό έχει συγκεκριμένο όνομα-ρόλο, ο οποίος υποδηλώνει και τις ιδιότητες που έχει. Συνεπώς κάθε σύμβολο είναι μια κλάση (συνήθως abstract) και το αποτέλεσμα καθε ρουτίνας ειναι μια νέα κλάση η οποία κάνει "extend" την κλάση του συμβόλου. Επιπλέον δηλώνουμε την κλάση της οποίας instance είναι ο κόμβος που επιστρέφεται ως αποτέλεσμα, σε κάθε περίπτωση μη τερματικού. Τα μη τερματικά είναι:

```
//Non Terminals
non terminal Expression Expr;    //Expressions (εκφράσεις συχνά χρησιμοποιούμενες και ως παράμετροι)
non terminal Statement Stmt;    //Statements (βασικές προτάσεις π.χ. κλήση συνάρτησης ή ένα while )
non terminal CompoundStatement CompStmt;    //Compound Statement ( Ένα block κώδικα με brackets {} )
non terminal List<Statement> StmtList;              //Λίστα από Statements
non terminal List<Declaration> VarFuncDecl; //Λίστα από δηλώσεις συναρτήσεων και global μεταβλητών στο αρχείο
non terminal FunctionDeclaration FuncDecl;      //Δήλωση συνάρτησης
non terminal ParameterDeclaration ParamDecl;  //Παράμετρος σε δήλωση συνάρτησης
non terminal List<ParameterDeclaration> ParamDeclList;  //Λίστα από παραμέτρους σε δήλωση συνάρτησης
non terminal TypeSpec TypeSpec;  //Τύπος μεταβλητής ή τύπος επιστρεφόμενης τιμής από συνάρτηση
non terminal VariableDeclaration  VarDecl;                  //Δήλωση μεταβλητής
non terminal CompUnit CompUnit;                 //"Ξεκίνημα ανάλυσης", μη τερματικό - στόχος
non terminal FunctionCallExpression FuncCall;               //Κλήση συνάρτησης
non terminal CompoundAssign CompAssign;      //Αρχικοποίηση ή Ανάθεση
non terminal Assignment Assign;  //Απλή ανάθεση ή νέο αντικείμενο (με χρήση constructor method)
non terminal List<Expression> ParamList;          //Λίστα από παραμέτρους σε δήλωση συνάρτησης.
non terminal ExitStatement RetStmt;               //Return Statement
```

Στη συνέχεια δηλώνουμε τις ιδιότητες προτεραιότητας και προσεταιριστικότητας:

```
//Precedence and Associativity
precedence right NOT;
precedence right EQ;
precedence left AND, OR;
precedence left EQUAL, NOT_EQUAL;
precedence left LESS , GREATER, LESS_EQ, GREATER_EQ;
precedence left PLUS, MINUS;
precedence left MULTI, DIV, MOD;
precedence right UMINUS;
```

Το μη τερματικό που είναι και ο στόχος της shift-reduce συντακτικής ανάλυσης:

```
//Start or the goal non-terminal for Shift-Reduce parsing
start with CompUnit;
```

Στο κομμάτι της γραμματικής, κάνουμε οδηγούμενους από το συντακτικό ορισμούς. Για κάθε κανόνα παραγωγής, κάθε σύμβολο της δεξιάς πλευράς συνοδεύεται από την αντίστοιχη σημασιολογική ρουτίνα. Ανάλογα την περίπτωση, δημιουργείται νέο instance της επιθυμητής κλάσης, προστίθεται ένα instance που παράχθηκε από άλλο action στο νέο, ή προσθέτουμε ιδιότητες. Σκοπός είναι να κρατήσουμε μόνο τη χρήσιμη πληροφορία που έρχεται από τον συντακτικό αναλυτή, για να τη χρησιμοποιήσουμε στο επόμενο βήμα. Επιπλέον κρατάμε πληροφορία για το ποιό σημείο βρίσκεται κάθε σύμβολο (γραμμή και στήλη) στο αρχείο. Έχοντας όλα τα παραπάνω υπ'όψη η γραμματική που περιγράφει την γλώσσα μας είναι η εξής:

```
/* GRAMMAR */
CompUnit ::=
        {:
                RESULT = new CompUnit();
                RESULT.setLine(0);
                RESULT.setColumn(0);
        :}
        | VarFuncDecl:vfd
        {:
                RESULT = new CompUnit(vfd);
                RESULT.setLine(vfdleft);
                RESULT.setColumn(vfdright);
        :}
        ;


VarFuncDecl ::= VarDecl:vd
        {:
                RESULT = new ArrayList<Declaration>();
                RESULT.add(vd);
        :}
        | FuncDecl:fd
        {:
                RESULT = new ArrayList<Declaration>();
                RESULT.add(fd);
        :}
```

```
        | VarFuncDecl:vfd VarDecl:vd
        {:
                vfd.add(vd);
                RESULT = vfd;
        :}
        | VarFuncDecl:vfd FuncDecl:fd
        {:
                vfd.add(fd);
                RESULT = vfd;
        :};

VarDecl ::= TypeSpec:type IDENTIFIER:id SEMICOLON
        {:
        RESULT = new VariableDeclaration(false,new IdentifierExpression(id),type);
        RESULT.setLine(typeleft);
        RESULT.setColumn(typeright);
        :}
        | TypeSpec:type LBRACKET RBRACKET IDENTIFIER:id SEMICOLON
        {:
        RESULT = new VariableDeclaration(true,new IdentifierExpression(id),type);
        RESULT.setLine(typeleft);
        RESULT.setColumn(typeright);
        :}
        ;

FuncDecl ::= TypeSpec:type IDENTIFIER:id LPAREN ParamDeclList:pdl RPAREN CompStmt:cs
        {:
        RESULT = new FunctionDeclaration(type,new IdentifierExpression(id) ,pdl,cs);
        RESULT.setLine(typeleft);
        RESULT.setColumn(typeright);
        :}
        ;

ParamDeclList ::=
        | ParamDecl:p
        {:
                RESULT = new ArrayList<ParameterDeclaration>();
                RESULT.add(p);
        :}
        | ParamDeclList:pl COMMA ParamDecl:p
        {:
                pl.add(p);
                RESULT = pl;
        :}
        ;

TypeSpec ::= INT        {: RESULT = TypeSpec.INT; :}
        | FLOAT         {: RESULT = TypeSpec.FLOAT; :}
        | CHAR          {: RESULT = TypeSpec.CHAR; :}
        | VOID  {: RESULT = TypeSpec.VOID; :}
        ;
```

```
ParamDecl ::= TypeSpec:type IDENTIFIER:id
        {:
        RESULT = new ParameterDeclaration(false,new IdentifierExpression(id),type);
        RESULT.setLine(typeleft);
        RESULT.setColumn(typeright);
        :}
        | TypeSpec:type LBRACKET RBRACKET IDENTIFIER:id
        {:
        RESULT = new ParameterDeclaration(true,new IdentifierExpression(id),type);
        RESULT.setLine(typeleft);
        RESULT.setColumn(typeright);
        :}
        ;


CompStmt ::= LCURBRACKET:d RCURBRACKET
        {:
        RESULT = new CompoundStatement(null);
        RESULT.setLine(dleft);
        RESULT.setColumn(dright);
        :}
        | LCURBRACKET:d StmtList:stmtl RCURBRACKET
        {:
        RESULT = new CompoundStatement(stmtl);
        RESULT.setLine(dleft);
        RESULT.setColumn(dright);
        :}
        ;

StmtList ::= Stmt:s
        {:
                RESULT = new ArrayList<Statement>();
                RESULT.add(s);
        :}
        | StmtList:sl Stmt:s
        {:
                sl.add(s);
                RESULT = sl;
        :}
        ;

Stmt ::= VarDecl:vd
        {:
        RESULT = vd;
        :}
        | FuncCall:f SEMICOLON
        {:
        RESULT=new FunctionCallStatement(f);
        RESULT.setLine(fleft);
        RESULT.setColumn(fright);
```

```
:}
| CompAssign:c SEMICOLON
{:
RESULT=c;
:}
| PRINT:p LPAREN Expr:e RPAREN SEMICOLON
{:
RESULT = new PrintStatement(e);
RESULT.setLine(pleft);
RESULT.setColumn(pright);
:}
| WHILE:w LPAREN Expr:e RPAREN CompStmt:cs
{:
RESULT = new WhileStatement(e,cs);
RESULT.setLine(wleft);
RESULT.setColumn(wright);
:}
| DO:d CompStmt:cs WHILE LPAREN Expr:e RPAREN SEMICOLON
{:
RESULT = new DoWhileStatement(cs,e);
RESULT.setLine(dleft);
RESULT.setColumn(dright);
:}
| IF:i LPAREN Expr:e RPAREN CompStmt:cs
{:
RESULT = new IfStatement(e,cs);
RESULT.setLine(ileft);
RESULT.setColumn(iright);
:}
| IF:i LPAREN Expr:e RPAREN CompStmt:cs1 ELSE CompStmt:cs2
{:
RESULT = new IfElseStatement(e,cs1,cs2);
RESULT.setLine(ileft);
RESULT.setColumn(iright);
:}
| RetStmt:r
{:
RESULT = r;
:}
| BREAK:b SEMICOLON //Semantics: for while clauses only
{:
RESULT = new ExitStatement(ExitType.BREAK);
RESULT.setLine(bleft);
RESULT.setColumn(bright);
:}
| CONTINUE:c SEMICOLON //Semantics: for while clauses only
{:
RESULT = new ExitStatement(ExitType.CONTINUE);
RESULT.setLine(cleft);
RESULT.setColumn(cright);
:}
```

```
        | CompStmt:c
        {:
        RESULT =c;
        :}
        ;

RetStmt ::= RETURN:r SEMICOLON
        {:
        RESULT = new ExitStatement(ExitType.RETURN);
        RESULT.setLine(rleft);
        RESULT.setColumn(rright);
        :}
        | RETURN:r Expr:e SEMICOLON
        {:
        RESULT = new ExitStatement(ExitType.RETURN, e);
        RESULT.setLine(rleft);
        RESULT.setColumn(rright);
        :}
        ;

CompAssign ::= Assign:a
        {:
        RESULT = new CompoundAssign(a);
        RESULT.setLine(aleft);
        RESULT.setColumn(aright);
        :}
        | TypeSpec:t Assign:a
        {:
        //Handle new Array initialization.
        RESULT = new CompoundAssign(new VariableDeclaration( a.getIsArray(), a.getId(), t), a);
        RESULT.setLine(tleft);
        RESULT.setColumn(tright);
        :}
        | IDENTIFIER:id LBRACKET INTEGER_LITERAL:index RBRACKET EQ Expr:e
                //E.g. myArray[50] = 100;
        {:
        RESULT = new CompoundAssign(new ArrayIndexAssignment(new IdentifierExpression(id),
index,e));
        RESULT.setLine(idleft);
        RESULT.setColumn(idright);
        :}
        ;

Assign ::= IDENTIFIER:id EQ Expr:e //anything else... x = 0;
        {:
        RESULT = new BasicAssignment(new IdentifierExpression(id), e);
        RESULT.setLine(idleft);
        RESULT.setColumn(idright);
        :}
        | IDENTIFIER:id EQ NEW TypeSpec:t LBRACKET INTEGER_LITERAL:size RBRACKET
                //E.g pnt = new int[5];
```

```
        {:
        RESULT = new ArrayInitAssignment(new IdentifierExpression(id), t, size);
        RESULT.setLine(idleft);
        RESULT.setColumn(idright);
        :}
        ;


Expr ::= INTEGER_LITERAL:l
        {:
        RESULT = new IntegerLiteralExpression(l);
        RESULT.setLine(lleft);
        RESULT.setColumn(lright);
        :}
        | DOUBLE_LITERAL:l
        {:
        RESULT = new DoubleLiteralExpression(l);
        RESULT.setLine(lleft);
        RESULT.setColumn(lright);
        :}
        | STRING_LITERAL:s
        {:
        RESULT = new StringLiteralExpression(s);
        RESULT.setLine(sleft);
        RESULT.setColumn(sright);
        :}
        | CHARACTER_LITERAL:c
        {:
        RESULT = new CharacterLiteralExpression(c);
        RESULT.setLine(cleft);
        RESULT.setColumn(cright);
        :}
        | IDENTIFIER:id
        {:
        RESULT = new IdentifierExpression(id);
        RESULT.setLine(idleft);
        RESULT.setColumn(idright);
        :}
        | FuncCall:f // implement above
        {:
        RESULT=f;
        :}

        | LPAREN:lp Expr:e RPAREN
        {:
        RESULT = new ParenthesisExpression(e);
        RESULT.setLine(lpleft);
        RESULT.setColumn(lpright);
        :}
        | Expr:e1 PLUS Expr:e2
        {:
```

```
RESULT = new BinaryExpression(Operator.PLUS, e1, e2);
RESULT.setLine(e1left);
RESULT.setColumn(e1right);
:}
| Expr:e1 MINUS Expr:e2
{:
RESULT = new BinaryExpression(Operator.MINUS, e1, e2);
RESULT.setLine(e1left);
RESULT.setColumn(e1right);
:}
| Expr:e1 MOD Expr:e2
{:
RESULT = new BinaryExpression(Operator.MOD, e1, e2);
RESULT.setLine(e1left);
RESULT.setColumn(e1right);
:}
|Expr:e1 MULTI Expr:e2
{:
RESULT = new BinaryExpression(Operator.MULTIPLY, e1, e2);
RESULT.setLine(e1left);
RESULT.setColumn(e1right);
:}
| Expr:e1 DIV Expr:e2
{:
RESULT = new BinaryExpression(Operator.DIVISION, e1, e2);
RESULT.setLine(e1left);
RESULT.setColumn(e1right);
:}
| Expr:e1 EQUAL Expr:e2
{:
RESULT = new BinaryExpression(Operator.EQUAL, e1, e2);
RESULT.setLine(e1left);
RESULT.setColumn(e1right);
:}
| Expr:e1 NOT_EQUAL Expr:e2
{:
RESULT = new BinaryExpression(Operator.NOT_EQUAL, e1, e2);
RESULT.setLine(e1left);
RESULT.setColumn(e1right);
:}
| Expr:e1 LESS Expr:e2
{:
RESULT = new BinaryExpression(Operator.LESS, e1, e2);
RESULT.setLine(e1left);
RESULT.setColumn(e1right);
:}
| Expr:e1 LESS_EQ Expr:e2
{:
RESULT = new BinaryExpression(Operator.LESS_EQ, e1, e2);
RESULT.setLine(e1left);
RESULT.setColumn(e1right);
```

```
        :}
        | Expr:e1 GREATER Expr:e2
        {:
        RESULT = new BinaryExpression(Operator.GREATER, e1, e2);
        RESULT.setLine(e1left);
        RESULT.setColumn(e1right);
        :}
        | Expr:e1 GREATER_EQ Expr:e2
        {:
        RESULT = new BinaryExpression(Operator.GREATER_EQ, e1, e2);
        RESULT.setLine(e1left);
        RESULT.setColumn(e1right);
        :}
        | Expr:e1 OR Expr:e2
        {:
        RESULT = new BinaryExpression(Operator.OR, e1, e2);
        RESULT.setLine(e1left);
        RESULT.setColumn(e1right);
        :}
        | Expr:e1 AND Expr:e2
        {:
        RESULT = new BinaryExpression(Operator.AND, e1, e2);
        RESULT.setLine(e1left);
        RESULT.setColumn(e1right);
        :}
        | NOT:n Expr:e
        {:
        RESULT = new UnaryExpression(Operator.NOT, e);
        RESULT.setLine(nleft);
        RESULT.setColumn(nright);
        :}
        | UMINUS:m Expr:e
        {:
        RESULT = new UnaryExpression(Operator.MINUS, e);
        RESULT.setLine(mleft);
        RESULT.setColumn(mright);
        :}
        %prec UMINUS
        ;

FuncCall ::= IDENTIFIER:id LPAREN ParamList:pl RPAREN
        {:
        RESULT = new FunctionCallExpression(new IdentifierExpression(id),pl);
        RESULT.setLine(idleft);
        RESULT.setColumn(idright);
        :}
        ;

ParamList ::=
        | Expr:e
        {:
```

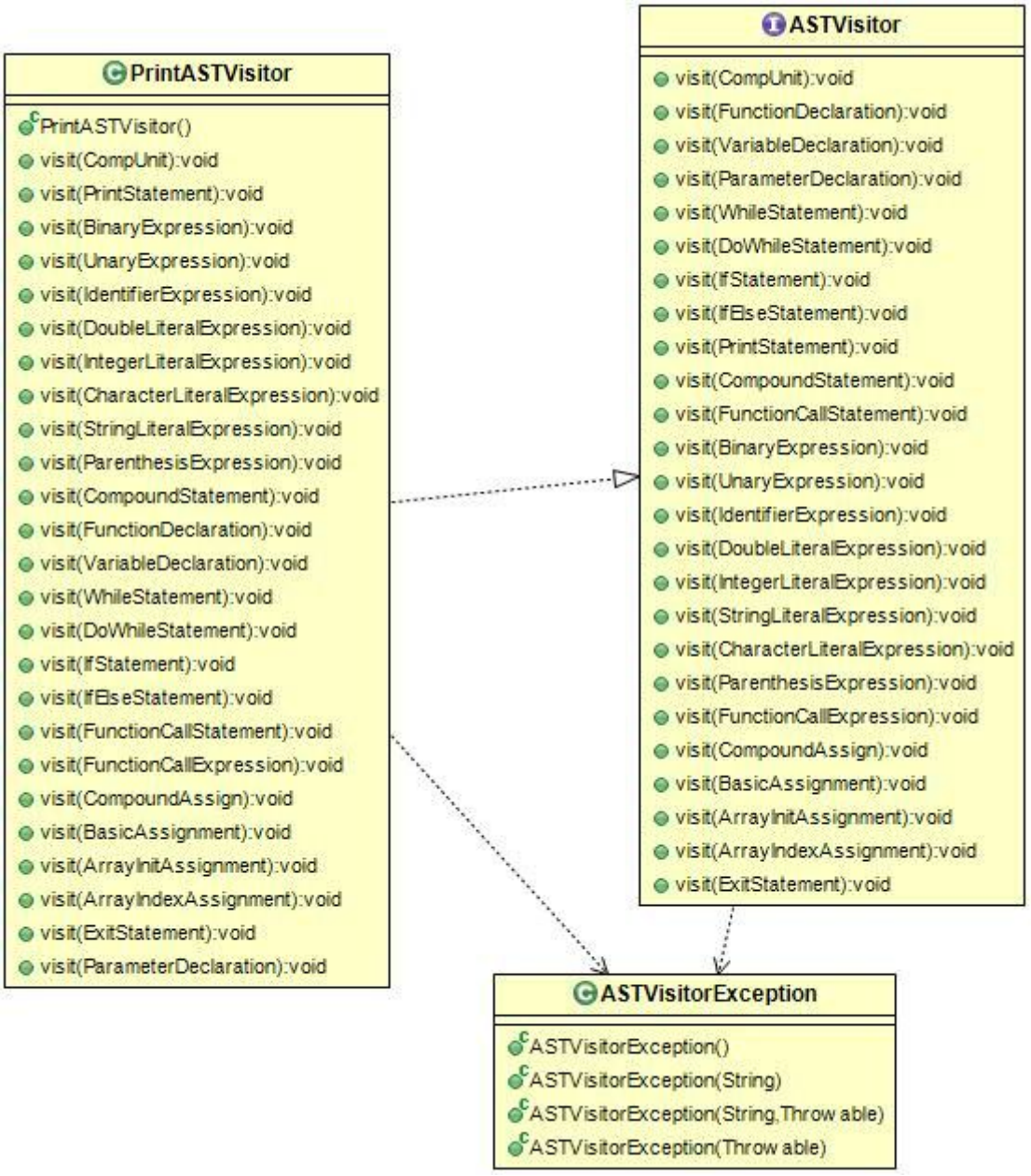
```
RESULT = new ArrayList<Expression>();
RESULT.add(e);
:}
| ParamList:pl COMMA Expr:e
{:
pl.add(e);
RESULT = pl;
:}
;
```

Κλάσεις για το Αφηρημένο Συντακτικό Δέντρο

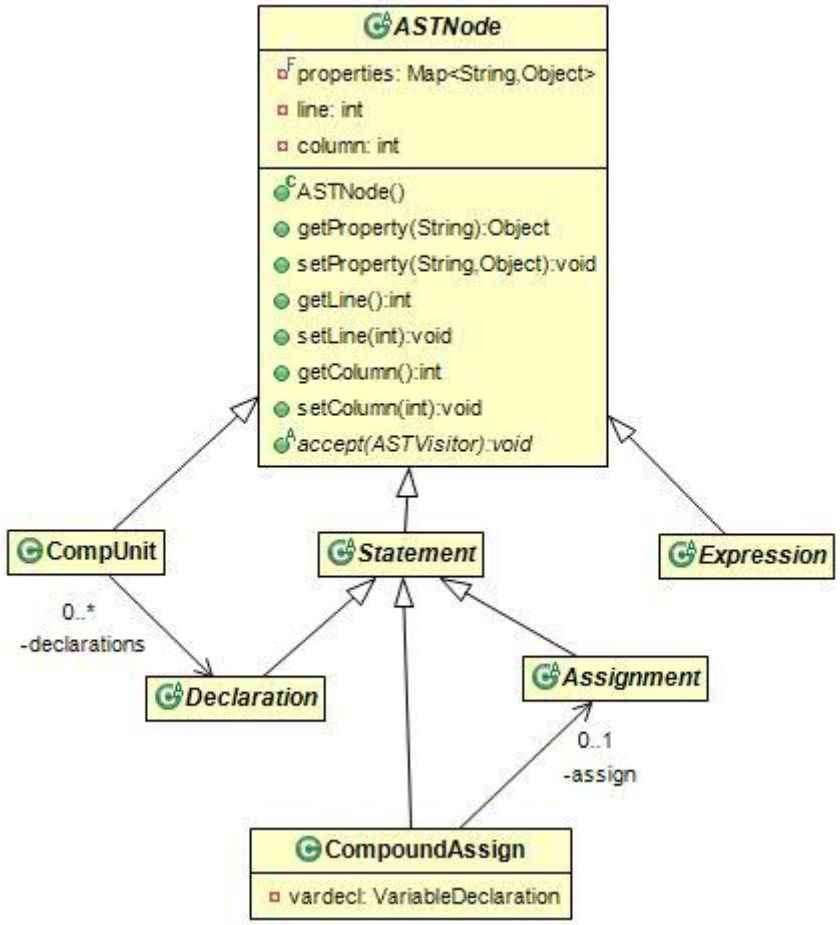
Στα παρακάτω διαγράμματα φαίνονται οι κλάσεις, καθώς και η κληρονομικότητα και χρήση στιγμιοτύπων της μιας από την άλλη.

Για την επεξεργασία της πληροφορίας που δημιουργείται από το συντακτικό δέντρο, χρησιμοποιούμε το visitor design pattern, με implementations του Visitor Interface να έχουν μια ρουτίνα για κάθε διαφορετικό τύπο κλάσης. Προς το παρόν η υλοποίηση που έχουμε μονάχα τυπώνει στην οθόνη τα αποτελέσματα για να ελέγξουμε την λειτουργικότητα.

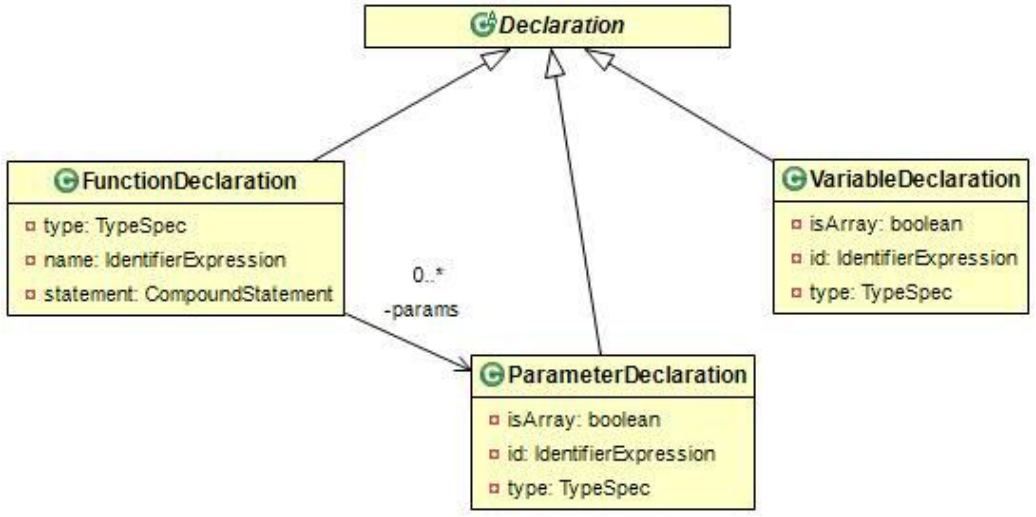
Παρακάτω φαίνεται το visitor και οι σχετικές κλάσεις.



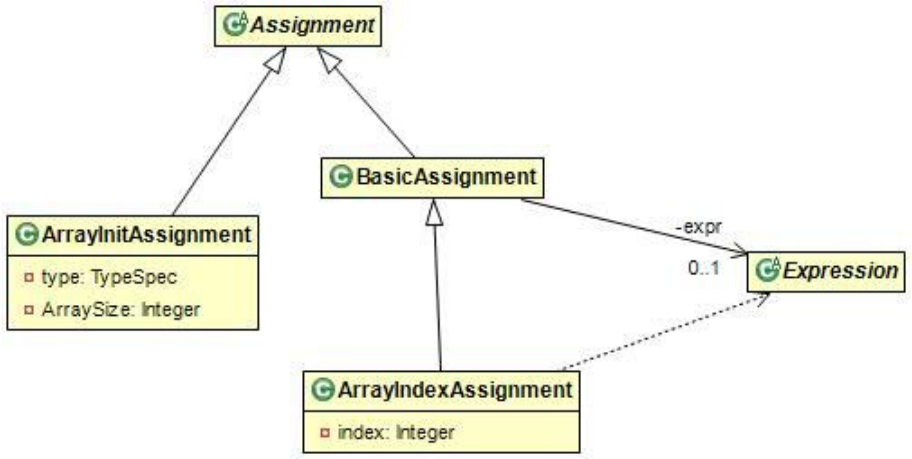
Τα “Visitables” είναι παιδιά της αφηρημένης κλάσης ASTNode, συνεπώς οι κόμβοι του δέντρου. Οι 2 πιο βασικές υποκλάσεις είναι Statements και Expressions. Οι σχέσεις των πιό βασικών υποκλάσεων της φαίνονται παρακάτω.



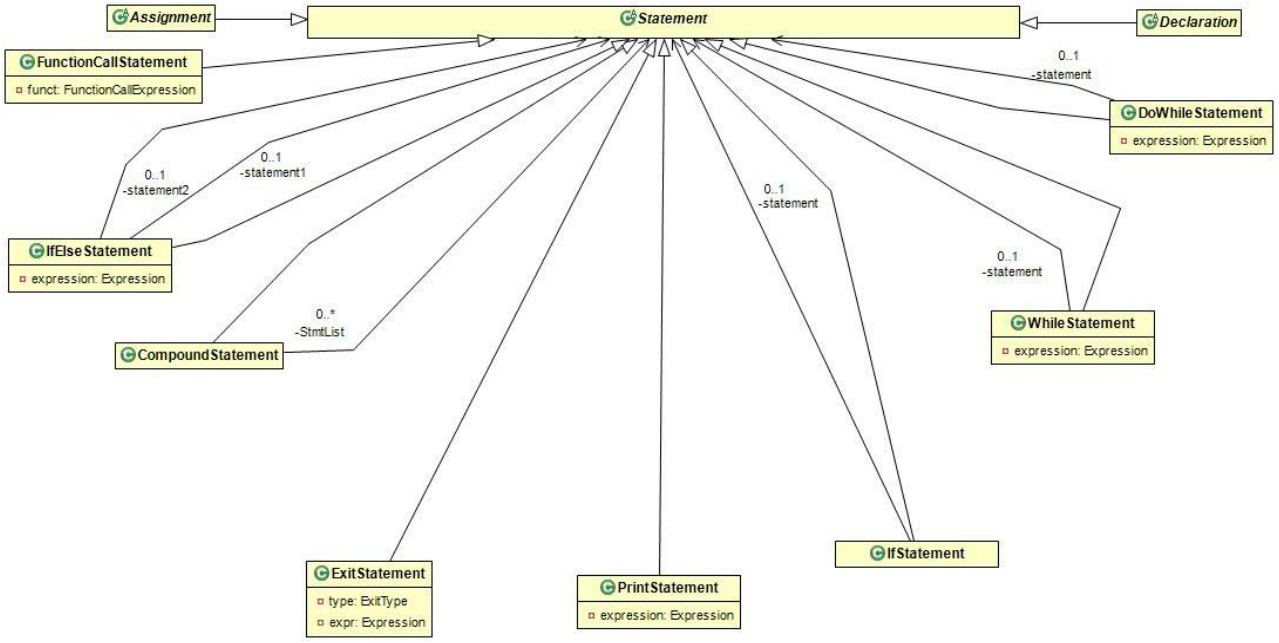
Το CompUnit είναι ο κόμβος-στόχος, ή ρίζα του δέντρου και αποτελείται από Declarations. Declarations και Assignments είναι οι βασικές υποκλάσεις του Statement. Το CompoundAssign(ment) είναι wrapper ενός assignment με ή χωρις την παρουσια ενός variable declaration.
Παρακάτω φαίνονται οι υποκλάσεις και οι σχέσεις μεταξύ των Declarations.



Τα Assignments είναι άλλη μια βασική κατηγορία Statement και φαίνονται παρακάτω.

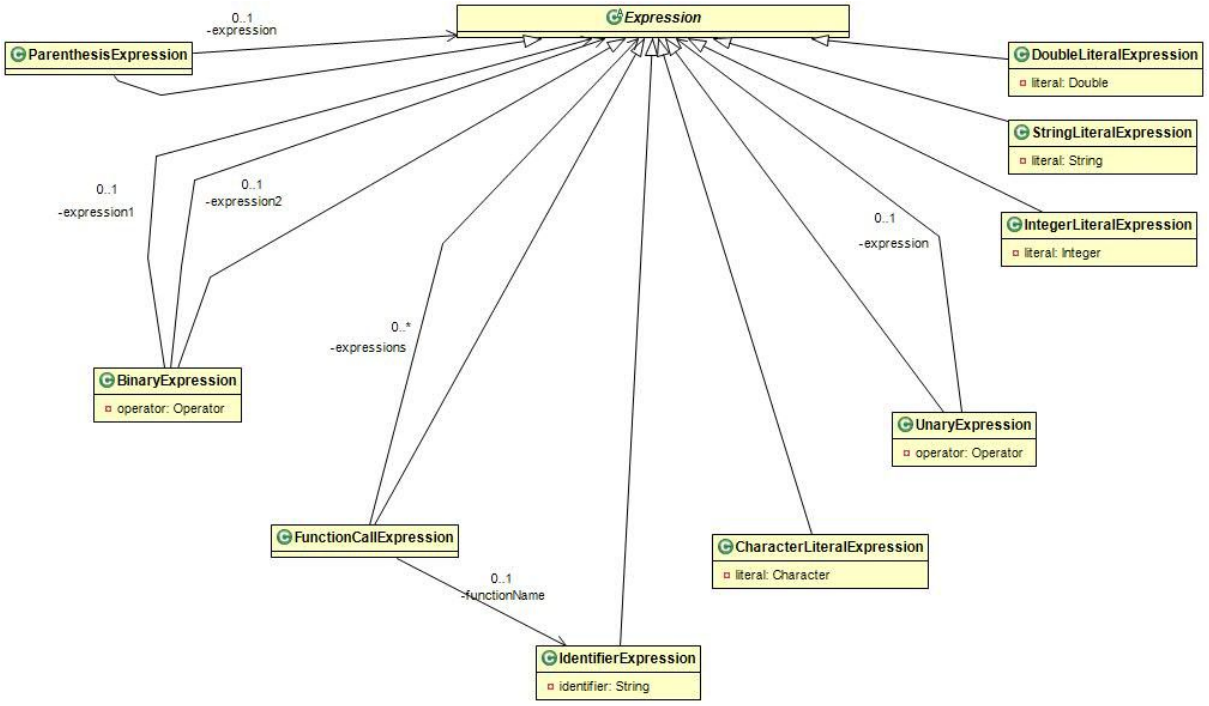


Τα statements είναι βασικό δομικό στοιχείο, όπου κάθε είδος χρειάζεται διαφορετική μεταχείριση από τον Visitor. Γι'αυτό και υπάρχουν πολλά είδη.
Όλες οι άμεσες υποκλάσεις του Statement και οι σχέσεις μεταξύ τους φαίνονται παρακάτω.



Επίσης υπάρχουν Statements που περιέχουν άλλα Statements, με κυριότερο το CompoundStatement.

Η abstract κλάση Expression και οι υποκλάσεις της φαίνονται παρακάτω.



Τα expressions είναι ακόμη ένα βασικό στοιχείο της γλώσσας και, όπως και τα statements, χρησιμοποιούνται και expressions μέσα σε expressions.

Για έλεγχο του project, στο αρχικό directory του project:
mvn package
java -jar target/compiler-0.0.3.jar input1.txt