ANTLR

ANother Tool for Language Recognition

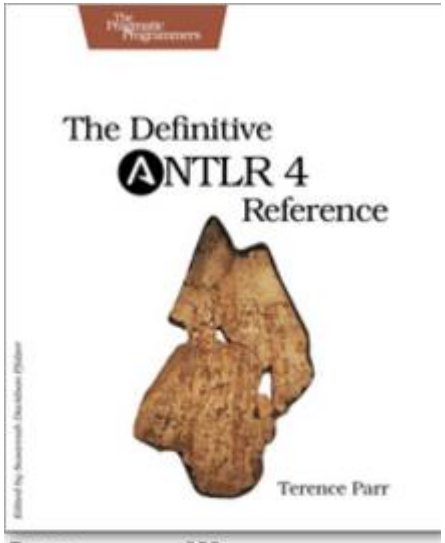
http://www.antlr.org/

What is ANTLR

- ANTLR (ANother Tool for Language Recognition) αποτελεί γεννήτορα συντακτικών αναλυτών για
  - ανάγνωση
  - επεξεργασία
  - εκτέλεση
  - μετατροπή δομημένου κειμένου ή αρχείων
- χρησιμοποιείται ευρέως για την κατασκευή μεταγλωττιστών και συναφών εργαλείων ανάπτυξης
- μετατρέπει μία γραμματική σε εκτελέσιμο κώδικα
- υποστηρίζει Java, C#, Python και άλλες γλώσσες προγραμματισμού

Behind ANTLR

- ο Terence Parr βρίσκεται πίσω από το ANTLR και δουλεύει πάνω σε τέτοια θέματα από το 1989.
- Είναι Καθηγητής Πληροφορικής στο University of San Francisco.



Pages	328
Published	2013-01-15
Release	P2.0 (2014-09-16)
ISBN	978-1-93435-699-9

An Example

```
grammar LabeledExpr;

prog: stat+;

stat:   expr NEWLINE                # printExpr
    |   ID '=' expr NEWLINE         # assign
    |   NEWLINE                     # blank
    ;

expr:   expr op=('*'|'/') expr      # MulDiv
    |   expr op=('+'|'-') expr      # AddSub
    |   INT                         # int
    |   ID                          # id
    |   '(' expr ')'                # parens
    ;

MUL:    '*';
DIV:    '/';
ADD:    '+';
SUB:    '-';

ID:         [a-zA-Z]+;
INT:        [0-9]+;
NEWLINE:    '\r'? '\n';
WS:         [ \t]+ -> skip;
```

- Ένα program (*prog*) αποτελείται από ένα ή περισσότερα *statements* (*stat*)
- Ένα *statement* μπορεί να είναι expression ή assignment.
- Ένα expression (*expr*) μπορεί να είναι mult/div, add/sub, a number, ένα ID ή ένα άλλο expression μέσα σε παρενθέσεις.
- Το *expr* είναι αναδρομικός κανόνας.

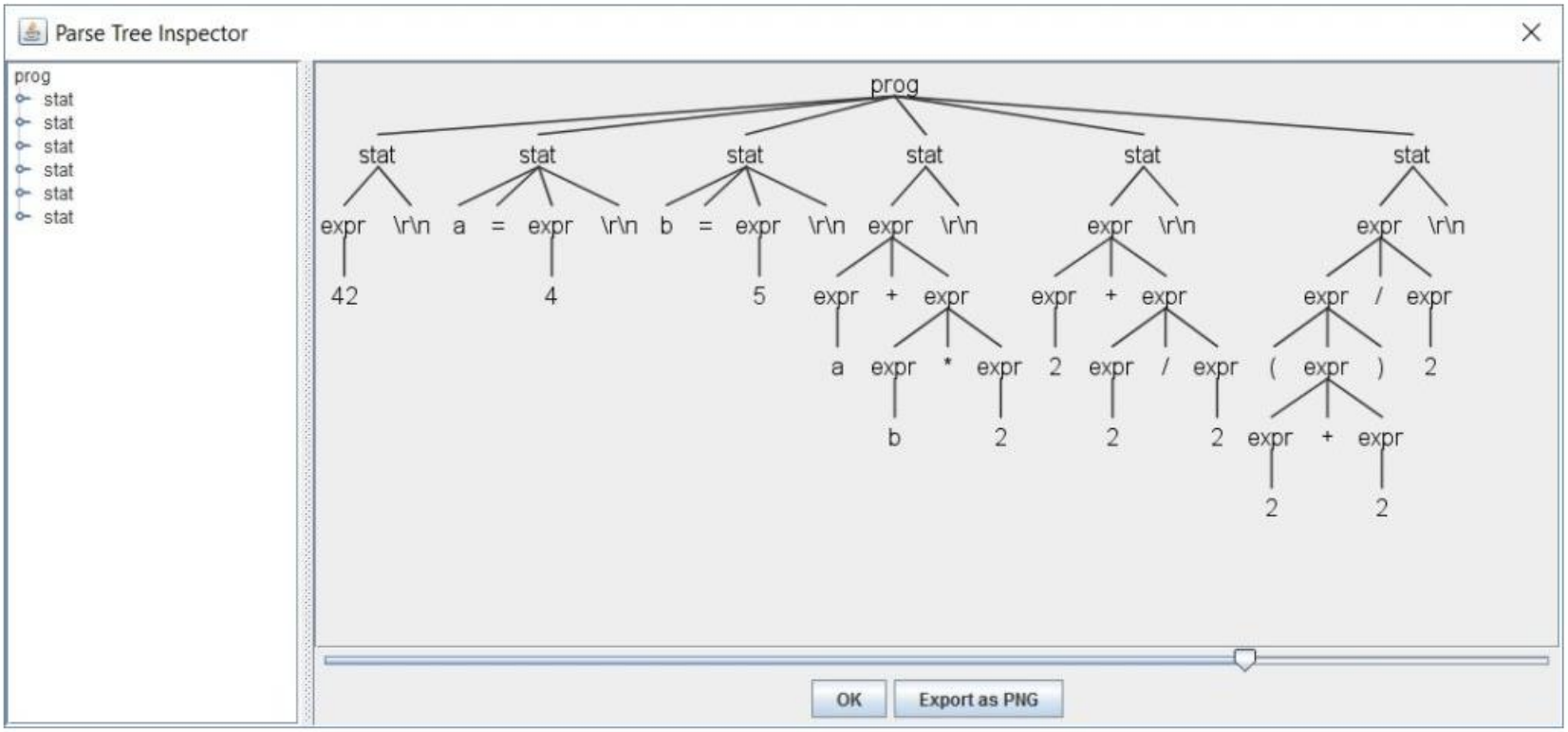
Compiling

- Το επόμενο βήμα είναι να μετατρέψουμε την γραμματική σε Java source code ως εξής
  - antlr LabeledExpr.g4
  - javac LabeledExpr*.java
- Το πρώτο μετατρέπει τη γραμματική σε Java source code.
- Το δεύτερο μεταγλωττίζει τον κώδικα Java ώστε να κάνει την γραμματική εκτελέσιμη

Testing

- grun LabeledExpr prog -gui t.expr

```
1  42
2  a=4
3  b=5
4  a+b*2
5  2+2/2
6  (2+2)/2
```



Comments

There are single-line, multiline, and Javadoc-style comments:

```
/** This grammar is an example illustrating the three kinds
 * of comments.
 */
grammar T;
/* a multi-line
  comment
*/

/** This rule matches a declarator for my language */
decl : ID ; // match a variable name
```

Small and Capital Letters

- Token names : ξεκινούν πάντοτε με κεφαλαίο γράμμα
- Parser rule names : ξεκινούν πάντοτε με μικρό γράμμα
- Ο αρχικός χαρακτήρας μπορεί να ακολουθείται από μικρά, κεφαλαία, ψηφία και κάτω παύλες.

```
ID, LPAREN, RIGHT_CURLY // token names/rules
expr, simpleDeclarator, d2, header_file // rule names
```

Literals

- Το ANTLR δεν διαχωρίζει ανάμεσα σε χαρακτήρες και συμβολοσειρές
- Όλες οι συμβολοσειρές με έναν ή περισσότερους χαρακτήρες τοποθετούνται μέσα σε απλά εισαγωγικά: πχ. ';', 'if', '>=', and '\''
- Το ANTLR επίσης καταλαβαίνει τους συνηθισμένους χαρακτήρες escape:

  '\n' (newline), '\r' (carriage return), '\t' (tab), '\b' (backspace)

Actions

- Τα actions είναι block κώδικα γραμμένα στην target language
- Μπορούν να τοποθετηθούν σε διάφορα σημεία της γραμματικής
- Σύνταξη: κείμενο μέσα σε άγκιστρα

```
my_list returns [IEvaluator e]
  : '$' LPAREN ops+=my_ident+ RPAREN { e = new MyListEvaluator(list_ops); }
  ;
```

```
multDivExpr returns [int value]
  : a = INT {$value = Int32.Parse($a.text);}
  ( '*' b = INT {$value *= Int32.Parse($b.text);}
  | '/' b = INT {$value /= Int32.Parse($b.text);})*;
```

Keywords

Here's a list of the reserved words in ANTLR grammars:

```
import, fragment, lexer, parser, grammar, returns,
locals, throws, catch, finally, mode, options, tokens
```

Grammar

```
/** Optional javadoc style comment */
grammar Name; ①
options {...}
import ... ;

tokens {...}
channels {...} // lexer only
@actionName {...}

rule1 // parser and lexer rules, possibly intermingled
...
ruleN
```

- Μπορούν να οριστούν options, imports, token specifications, and actions με οποιαδήποτε σειρά
- Μπορεί να υπάρχει το πολύ μία φορά κάποιο από τα: options, imports, and token specifications
- Απαιτείται τουλάχιστον ένα rule
- Το όνομα του αρχείου που περιέχει τη γραμματική X πρέπει να είναι X.g4

Rules

```
ruleName : alternative1 | ... | alternativeN ;
```

- Τα Parser rules πρέπει να ξεκινάνε με μικρό γράμμα
- Τα Lexer rules πρέπει να ξεκινάνε με κεφαλαίο γράμμα

Parser και Lexer Grammars

- Οι γραμματικές στις οποίες το header δεν δηλώνεται κάτι πέρα από το όνομα της γραμματικής είναι combined grammars και μπορεί να περιέχουν και lexical και parser rules.
- Όταν μία γραμματική θέλουμε να περιέχει μόνο parser rules, χρησιμοποιούμε το ακόλουθο header:

```
parser grammar Name;
...
```

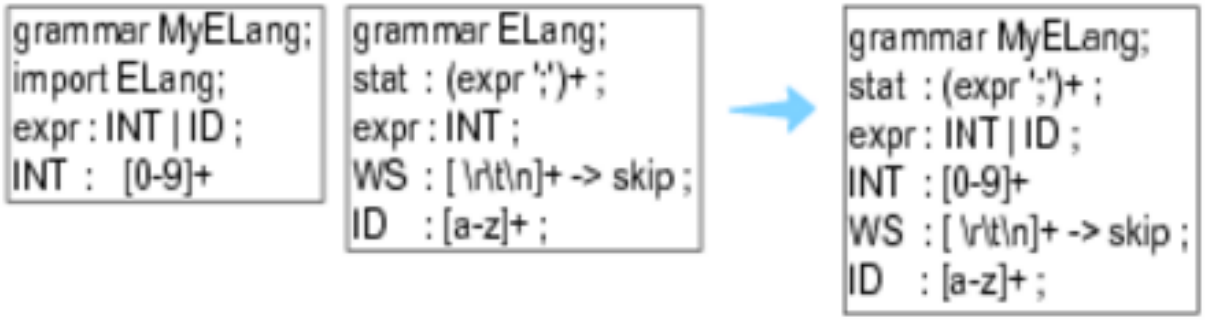
- ενώ για μία lexer grammar:

```
lexer grammar Name;
...
```

Grammar Imports

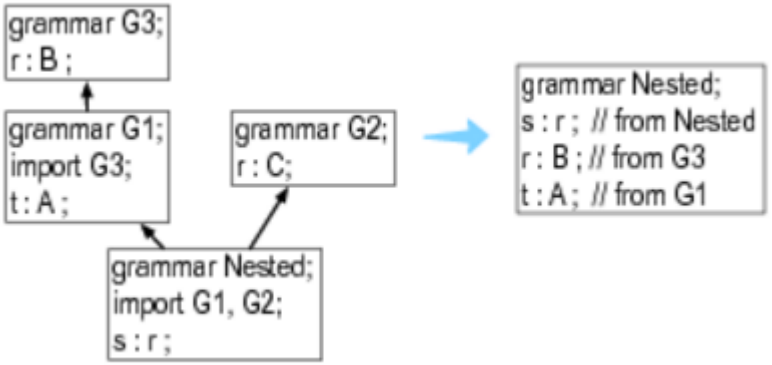
- Ο μηχανισμός grammar imports επιτρέπει να τμηματοποιήσουμε μία γραμματική σε λογικά και επαναχρησιμοποιήσιμα τμήματα
- Το ANTLR χρησιμοποιεί τον μηχανισμό αυτόν όπως γίνεται στις object-oriented γλώσσες προγραμματισμού με τις υπερκλάσεις
- Μία γραμματική κληρονομεί όλα τα rules, tokens specifications, and named actions από την εισαγόμενη γραμματική
- Οι κανόνες στην κύρια γραμματική υποσκελίζουν (override) τους κανόνες από την εισαγόμενη γραμματική
- Το αποτέλεσμα όλων των imports είναι μία combined grammar

Grammar Imports



- Το MyELang κληρονομεί τα rules stat, WS, και ID, αλλά υποσκελίζει τα rule expr ενώ προσθέτει το INT

Grammar Imports



- Οι imported grammars μπορούν να κάνουν import άλλες γραμματικές
- εάν δύο ή περισσότερες imported grammars ορίζουν το κανόνα r, το ANTLR θα διαλέξει τον πρώτο r που θα βρει.
- Στο σχήμα, το ANTLR εξετάζει τις γραμματικές με την ακόλουθη σειρά: G1, G3, G2

Grammar Imports

- Δεν μπορεί οποιαδήποτε γραμματική να κάνει import οποιαδήποτε άλλη γραμματική:
  - Οι lexer grammars κάνουν import lexers.
  - Οι parsers κάνουν import parsers.
  - Οι combined grammars κάνουν import lexers ήparsers.

Tokens

```
tokens { Token1, ..., TokenN }
```

- έμμεσοι ορισμοί - προειδοποίηση

```
$ cat Tok.g4
grammar Tok;
tokens { A, B, C }
a : X ;
$ antlr4 Tok.g4
warning(125): Tok.g4:3:4: implicit definition of token X in parser
$ cat Tok.tokens
A=1
B=2
C=3
X=4
```

Actions at the Grammar Level

```
grammar Count;

@header {
package foo;
}

@members {
int count = 0;
}

list
@after {System.out.println(count+" ints");}
: INT {count++;} (',' INT {count++;} )*
;

INT : [0-9]+ ;
WS : [ \r\t\n]+ -> skip ;
```

- Υπάρχουν μόνο δύο είδη από named actions που ορίζονται έξω από τη γραμματική: είναι τα header και τα members.
- το πρώτο εισάγει κώδικα πριν τον κώδικα της παραγόμενης κλάσης
- το δεύτερο εισάγει κώδικα μέσα στην παραγόμενη κλάση σαν fields και methods.

Parser Rules

- Οι Parsers αποτελούνται από ένα σύνολο από parser rules είτε σε μία parser είτε σε μία combined grammar
- Μία εφαρμογή Java εκκινεί τον parser καλώντας τον αρχικό κανόνα
- Ο πιο απλός κανόνας είναι ένας κανόνας με μία και μοναδική επιλογή που τερματίζεται από ένα ερωτηματικό (ελληνικό).

```
/** Javadoc comment can precede rule */
retstat : 'return' expr ';' ;
```

Parser Rules

- Οι κανόνες μπορεί να δίνουν εναλλακτικές επιλογές διαχωριζόμενες από |
- Οι επιλογές αυτές μπορεί να είναι μία λίστα από rule elements ή να είναι και άδεια

```
operator:
    stat: retstat
    | 'break' ';'
    | 'continue' ';'
    ;
```

```
superClass
    : 'extends' ID
    | // empty means other alternative(s) are optional
    ;
```

Actions and Attributes

- Τα Actions είναι block κειμένου γραμμένα στην target language τα οποία τοποθετούμε μέσα σε άγκιστρα
- Αυτά ενεργοποιούνται ανάλογα με τη θέση τους στη γραμματική
- Για παράδειγμα, ο ακόλουθος κανόνας ΄τυπώνει "found a decl" αφού ο συντακτικός αναλυτής έχει βρει μία δήλωση:

```
decl: type ID ';' {System.out.println("found a decl");} ;
type: 'int' | 'float' ;
```

Actions and Attributes

- Συχνά τα actions προσπελαύνουν τα attributes των tokens και των rules:

```
decl: type ID ';'
      {System.out.println("var "+$ID.text+":"+$type.text+";");}
    | t=ID id=ID ';'
      {System.out.println("var "+$id.text+":"+$t.text+";");}
    ;
```

Token Attributes

- Όλα τα token έχουν μία συλλογή από predefined, read-only attributes.
- Τα attributes περιέχουν χρήσιμες ιδιότητες όπως τα token type και text.
- Τα actions έχουν πρόσβαση σε αυτά τα attributes χρησιμοποιώντας τον συμβολισμό: $label.attribute, όπου το label δείχνει ένα συγκεκριμένο στιγμιότυπο από το token (τα a και b στο παράδειγμα παρακάτω χρησιμοποιούνται μέσα στο action ως $a and $b).
- Συχνά ένα token αναφέρεται μόνο μία φορά στον κανόνα. Στην περίπτωση αυτή το όνομα του token μπορεί να χρησιμοποιηθεί μέσα στο action χωρίς να υπάρχει αμφισημία (το token INT μπορεί να χρησιμοποιηθεί σαν $INT στο action).

```
r : INT {int x = $INT.line;}
    ( ID {if ($INT.line == $ID.line) ...;} )?
    a=FLOAT b=FLOAT {if ($a.line == $b.line) ...;}
  ;
```

Token Attributes

- Παρακάτω υπάρχουν δύο αναφορές στο token FLOAT
- άρα το να χρησιμοποιήσουμε το $FLOAT περιέχει αμφισημία
- πρέπει να χρησιμοποιήσουμε labels για να καθορίσουμε το token στο οποίο αναφερόμαστε

```
r : INT {int x = $INT.line;}
    ( ID {if ($INT.line == $ID.line) ...;} )?
    a=FLOAT b=FLOAT {if ($a.line == $b.line) ...;}
  ;
```

Token Attributes

- Οι αναφορές σε token που βρίσκονται σε δύο εναλλακτικές ενός κανόνα μπορούν να θεωρηθούν μοναδικές, αφού μόνο η μία από τις δύο επιλογές θα ενεργοποιηθεί.
- για παράδειγμα, στον ακόλουθο κανόνα, τα actions και στις δύο επιλογές μπορούν να χρησιμοποιήσουν το $ID απευθείας, χωρίς label:

```
r : ... ID {System.out.println($ID.text);}
| ... ID {System.out.println($ID.text);}
;
```

Token Attributes

- για να προσπελάσουμε tokens σε σταθερές, χρειαζόμαστε label:

```
stat: r='return' expr ';' {System.out.println("line="+$r.line);} ;
```

- μπορούμε να το χρησιμοποιήσουμε και για να δούμε αν ένας προαιρετικός κανόνας πρέπει να ενεργοποιηθεί:

```
stat: 'if' expr 'then' stat (el='else' stat)?
{if ( $el!=null ) System.out.println("found an else");}
| ...
;
```

Token Attributes

text	String	The text matched for the token; translates to a call to getText. Example: $ID.text.
type	int	The token type (nonzero positive integer) of the token such as INT; translates to a call to getType. Example: $ID.type.
line	int	The line number on which the token occurs, counting from 1; translates to a call to getLine. Example: $ID.line.
pos	int	The character position within the line at which the token's first character occurs counting from zero; translates to a call togetCharPositionInLine. Example: $ID.pos.
index	int	The overall index of this token in the token stream, counting from zero; translates to a call to getTokenIndex. Example: $ID.index.
int	int	The integer value of the text held by this token; it assumes that the text is a valid numeric string. Handy for building calculators and so on. Translates to Integer.valueOf(text-of-token). Example: $INT.int.

Parser Rule Attributes

- Το ANTLR ορίζει κάποια read-only attributes συνδυασμένα με έναν κανόνα του parser ρκαι τα οποία είναι διαθέσιμα στα actions.
- Φυσικά τα actions μπορούν να προσπελάσουν attributes που έχουν πάρει τιμή νωρίτερα στον κώδικα μόνο.
- Η σύνταξη είναι $r.attr για τον κανόνα r ή για κάποια ετικέτα που αντιστοιχεί σε κανόνα.
- Για παράδειγμα, το $expr.text επιστρέφει σαν κείμενο ό,τι έχει αναγνωριστεί από το την τελευταία ενεργοποίηση του κανόνα expr:

```
returnStat : 'return' expr {System.out.println("matched "+$expr.text);} ;
```

or

```
returnStat : 'return' e=expr {System.out.println("matched "+e.text);} ;
```

Parser Rule Attributes

text	String	The text matched for a rule or the text matched from the start of the rule up until the point of the `$text` expression evaluation. Note that this includes the text for all tokens including those on hidden channels, which is what you want because usually that has all the whitespace and comments. When referring to the current rule, this attribute is available in any action including any exception actions.
start	Token	The first token to be potentially matched by the rule that is on the main token channel; in other words, this attribute is never a hidden token. For rules that end up matching no tokens, this attribute points at the first token that could have been matched by this rule. When referring to the current rule, this attribute is available to any action within the rule.
stop	Token	The last nonhidden channel token to be matched by the rule. When referring to the current rule, this attribute is available only to the after and finally actions.
ctx	ParserRuleContext	The rule context object associated with a rule invocation. All of the other attributes are available through this attribute. For example, `$ctx.start` accesses the start field within the current rules context object. It's the same as `$start`.

Lexer Rules

- μία γραμματική λεκτικού αναλυτή απαρτίζεται από lexer rules
- προαιρετικά μπορεί να αποτελείται από πολλαπλά modes.
- Τα modes μας επιτρέπουν να χωρίζουμε έναν lexer σε πολλούς sub-lexers.
- Οι κανόνες του Lexer ορίζουν token
- Οι κανόνες ενός Lexer πρέπει να ξεκινούν με κεφαλαίο γράμμα, κάτι που τους διαχωρίζει από αυτούς του parser

Lexer Modes

- Τα modes επιτρέπουν ομαδοποίηση λεκτικ΄ν κανόνων με βάση το περιεχόμενο
- Οι Lexers ξεκινούν στο default mode
- Όλοι οι κανόνες είναι στο default mode εκτός αν οριστεί κάποια εντολή mode.

```
rules in default mode
...
mode MODE1;
rules in MODE1
...
mode MODEN;
rules in MODEN
...
```

Lexer Modes

```
STRING : [a-z-]+;
LBRACK : '[' -> pushMode(CharSet);

mode CharSet;

DASH : '-';
NUMBER : [0-9]+;
RBRACK : ']' -> popMode;
```

- Μολίς συναντήσουμε ένα '[', ο lexer θα μεταβεί στο mode CharSet υέως ότου ο χαρακτήρας ']' εμφανιστεί και η popMode εκτελεστεί.

Recursive Lexer Rules

- Επιτρέπονται αναδρομικοί ορισμοί, κάτι που είναι ασυνήθιστο σε εργαλεία λεκτικής ανάλυσης.
- είναι όμως χρήσιμο για φωλιασμένες δομές, π.χ.: {...{...}...}

```
lexer grammar Recur;

ACTION : '{' ( ACTION | ~[{}] )* '}' ;

WS : [ \r\t\n]+ -> skip ;
```

Calculator

```
grammar Calculator;
INT    : [0-9]+;
DOUBLE : [0-9]+'.'[0-9]+;
PI     : 'pi';
E      : 'e';
POW    : '^';
NL     : '\n';
WS     : [ \t\r]+ -> skip;
ID     : [a-zA-Z_][a-zA-Z_0-9]*;

PLUS  : '+';
EQUAL : '=';
MINUS : '-';
MULT  : '*';
DIV   : '/';
LPAR  : '(';
RPAR  : ')';
```

Calculator

```
input
    : setVar NL input     # ToSetVar
    | plusOrMinus NL? EOF # Calculate
    ;

setVar
    : ID EQUAL plusOrMinus # SetVariable
    ;

plusOrMinus
    : plusOrMinus PLUS multOrDiv  # Plus
    | plusOrMinus MINUS multOrDiv # Minus
    | multOrDiv                   # ToMultOrDiv
    ;

multOrDiv
    : multOrDiv MULT pow # Multiplication
    | multOrDiv DIV pow  # Division
    | pow                # ToPow
    ;
```

```
pow
    : unaryMinus (POW pow)? # Power
    ;

unaryMinus
    : MINUS unaryMinus # ChangeSign
    | atom             # ToAtom
    ;

atom
    : PI                    # ConstantPI
    | E                     # ConstantE
    | DOUBLE                # Double
    | INT                   # Int
    | ID                    # Variable
    | LPAR plusOrMinus RPAR # Braces
    ;
```

Calculator

Getting started

1. Install ANTLR v4 (manual)
2. Generate ANTLR files `antlr4 Calculator.g4 -no-listener -visitor -o app`
3. Copy visitor implementation `cp *.java app`
4. Compile `javac app/*.java`

Run

Type in console `cd app && java Run`

```
a = 1+2
b = a^2
c = a + b * (a - 1)
a + b + c
```

Listeners

```
grammar T;
stat: 'return' e ';' # Return
    | 'break' ';' # Break
    ;
e   : e '*' e # Mult
    | e '+' e # Add
    | INT # Int
    ;
```

```
public interface AListener extends ParseTreeListener {
    void enterReturn(AParser.ReturnContext ctx);
    void exitReturn(AParser.ReturnContext ctx);
    void enterBreak(AParser.BreakContext ctx);
    void exitBreak(AParser.BreakContext ctx);
    void enterMult(AParser.MultContext ctx);
    void exitMult(AParser.MultContext ctx);
    void enterAdd(AParser.AddContext ctx);
    void exitAdd(AParser.AddContext ctx);
    void enterInt(AParser.IntContext ctx);
    void exitInt(AParser.IntContext ctx);
}
```

Listeners

- Reuse of the same labels. The grammar:

```
e : e '*' e # BinaryOp
| e '+' e # BinaryOp
| INT # Int
;
```

- would generate the following listener methods for e:

```
void enterBinaryOp(AParser.BinaryOpContext ctx);
void exitBinaryOp(AParser.BinaryOpContext ctx);
void enterInt(AParser.IntContext ctx);
void exitInt(AParser.IntContext ctx);
```

Ευχαριστώ