ANTLR-1

COM S 319

ANTLR

http://www.antlr.org

ANTLR (ANother Tool for Language Recognition) is a powerful **parser generator** for reading, processing, executing, or translating structured text or binary files. It's widely used to build languages, tools, and frameworks. From a grammar, ANTLR generates a parser that can build and walk parse trees.

ANTLR

- ANTLR
  - completely Java code.
  - Both lexer and parser rules are specified in one file.

- Antlr Commands

```
// generates java code for lexer/parser
antlr4 Expr.g4
javac Expr*.java        // compile the code
// if just tokenizing then grun Expr tokens < input_file

// if parsing then  grun Expr prog –gui (or –tree)
100+2*34 ^D
```

Only Lexer

```
lexer grammar ABC;

options
{  // antlr will generate java lexer and parser
  language = Java;
}

// follow with multiple lexer rules
//the grammar must contain at least one lexer rule
RULE_NAME_IN CAPS: regular expression;
SALUTATION: ('Hello world');
ENDSYMBOL: '!' ;
```

LEXER RULES

- RULE_NAME : RULE_CONTENTS ;

character	meaning	example	matches
|	logical *OR*	`'a' | 'b'`	either `'a'` or `'b'`
?	optional	`'a' 'b'?`	either `'ab'` or `'a'`
*	none or more	`'a'*`	nothing, `'a'`, `'aa'`, `'aaa'`, ...
+	once or more	`'a'+`	`'a'`, `'aa'`, `'aaa'`, ...
~	negation	`~('a' | 'b')`	any character (in the range \u0000..\uFFFF) except `'a'` and `'b'`
(...)	grouping	`('a' 'b')+`	`'ab'`, `'abab'`, `'ababab'`, ...

Example: ABC.g4

```
lexer grammar ABC;

options
{
  // antlr will generate java lexer and parser
  language = Java;
}

// multiple lexer rules
//the grammar must contain at least one lexer rule
SALUTATION: ('Hello world');
ENDSYMBOL: '!' ;
```

rules can have code attached

RULENAME: regexpression { code } ;

Example:

HELLO: 'hello' { System.out.println("matching HELLO rule:" + getText());};

- getText() returns token.
- skip() consumes token and skips it.

fragment

fragment RULENAME: regexpression ;

Example:

```
fragment DIGIT: [0123456789];
fragment ALPHA: [a-zA-Z] ;
```

- fragments cannot have code attached
- fragments can be used to make up complex regexs

Example

```
WORD: ALPHA ( ALPHA | DIGIT )*
    {System.out.println("found word: " + getText()); };
```

greedy vs non greedy

consider line
/* hello */ /* this is an example */

// what does this match?
COMMENT1 : '/*' .* '*/' -> skip ;

// what does this match?
COMMENT2: '/*' .*? '*/' -> skip ;

// add ? to * or + or ? to get non-greedy match