ANTLR4

Tips and Tricks

Greta Yorsh

LEXER

Simple lexer

- start with a capital letter

- priority over ID

- fragment of lexeme
- not a token
- not passed to parser
- simplify the grammar

- catch all unmatched characters and pass to the parser for error handling

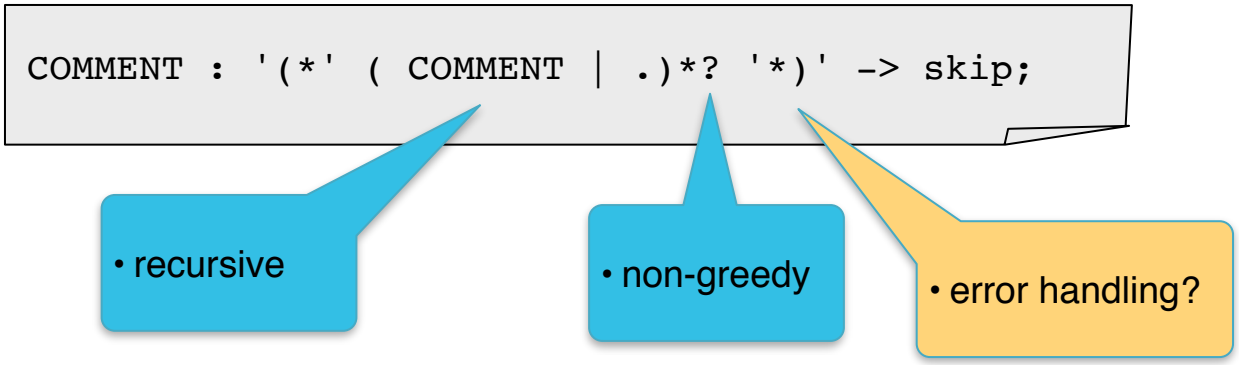
```
lexer grammar ExampleLexer;

INT       : DIGIT+ ;
fragment
DIGIT     : [0-9] ;

WHILE     : [wW][hH][iI][lL][eE] ;

ID        :  [a-z] IDENTIFIER* ;
fragment
LETTER    :  [a-zA-Z];

COMMENT   : '/*' .*? '*/'    -> skip;

WHITESPACE :
(' ' | '\n' | '\r' | '\t' | '\u000B')+ -> skip;

ERROR     : . ;
```

- greedy operator + matches as much input as possible

- non-greedy operator *? matches until first end-of-comment

- lexer command skip
- token not passed to parser

Nested block comments



Lexer modes: nested block comments

- group lexical rules by context
- must separate parser rules to another file

```
BEGIN_COMMENT: '(*' -> skip, pushMode(COMMENT_MODE);

mode COMMENT_MODE;

END_COMMENT: '*)' -> skip, popMode;

COMMENT_TEXT: . -> skip;

mode DEFAULT_MODE;
    ....
```

- mini-lexer for nested comments

Lexer modes: strings

```
BEGIN_STRING  : '"' -> pushMode(STRING_MODE);

mode STRING_MODE;

STRING_TEXT   : ~[\\\r\n"];

STRING_ESCAPE : '\\' [trn"'\\];

END_STRING    : '"' -> popMode;
```

- mini-lexer for strings

- error handling?

Lexer modes: strings

```
BEGIN_STRING  : '"' -> pushMode(STRING_MODE);

mode STRING_MODE;

STRING_TEXT   : ~[\\\r\n"];

STRING_ESCAPE : '\\' [trn"'\\];

END_STRING    : '"' -> popMode;

UNTERMINATED_STRING : '\n'
    { setText("Unterminated string constant"); }
     -> type(ERROR), popMode;
```

- change text associated with token
- action
- change type of token
- command

Lexer members: strings

```
@lexer::members {
    StringBuilder buf;
}
...
BEGIN_STRING
  : '"' { buf = new StringBuilder(); }
    -> skip, pushMode(STRING_MODE)
  ;
mode STRING_MODE;
  STRING_TEXT
    : ~[\\\r\n"]
    { (buf.length() < StringTable.MAXSIZE) }?
    { buf.append(getText()); }
  ;
  STRING_ESCAPE_TAB
    : '\\' [t] { buf.append("\t"); }
    ;
  END_STRING
    : '"' -> skip, popMode
    ;
 ....
```

- "target-language" specific code in Java
- field of lexer class

- action

- predicate

- action

- NOT COMPLETE

Lexer

- fragment
- skip
- modes
- actions
- commands
- whitespace characters
- escape characters
- error token

PARSER

Direct recursion only

```
parser grammar Example;

expr : ID | logical_expr | bitwise_expr ;

logical_expr  : expr AND expr | expr OR expr;

bitwise_expr  : expr BIT_AND expr | expr BIT_OR expr;
```

- indirect recursion: antlr error

```
parser grammar Example;

expr : ID
     | expr AND expr | expr OR expr
     | expr BIT_AND expr | expr BIT_OR expr;
```

Associativity

- default is left-associative

- assignment operator is right-associative

```
e : e '*' e
  | e '+' e
  |<assoc=right> e ':=' e
  | INT
  ;
....
```

- which operators are non-associative?
- how to handle it with Antlr?

Labelled Alternatives

```
parser grammar Example;

e   : e '*' e   # Mult
    | e '+' e   # Add
    | INT       # Int
    ;
```

- specialize nodes in parse tree
- all alternatives must be labeled

Error token

```
parser grammar CoolParser;
....

error : ERROR
    { Utilities.lexError(); }
    ;
```

Parser

- mutual recursive rules
- non-associative operators
- labeled alternatives
- to lex or to parse?

Lexer

	Status
Keywords	
Whitespace	
Object identifiers	
Type identifiers	
Integers	
Strings	
Escape characters	
Line comment	
Block comment	

Parser

	Status
Class	
Method	
Attribute	
Formal	
Assign	
Const	
Binop	
Unop	
Conditional	
Loop	
Let	
Case	

Examples

	Lexer	Parser	AST construction
hello_world.cl			
factorial.cl			
cool.cl			
arith.cl			