

Desarrollo de Lenguajes Orientados a Objeto y Compiladores Avanzados [MII-779]

Capítulo 2: Análisis Léxico

Dr. Ricardo Soto

[ricardo.soto@ucv.cl]

[<http://www.inf.ucv.cl/~rsoto>]

Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

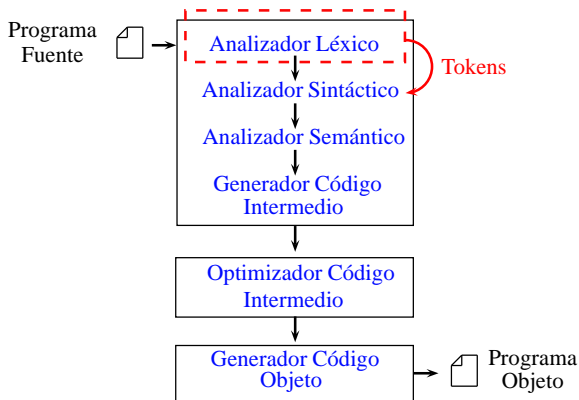


PONTIFICIA UNIVERSIDAD
CATOLICA
DE VALPARAISO



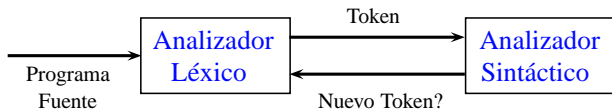
1. Introducción

El **análisis léxico** corresponde a la primera fase de un compilador. Es la encargada de recibir el programa fuente y reconocer los **tokens**.



2. Funciones del Analizador Léxico

- **Generar** una lista ordenada de **tokens** a partir de los caracteres de entrada
- **Interactuar** con el analizador sintáctico, enviándole los **tokens** generados



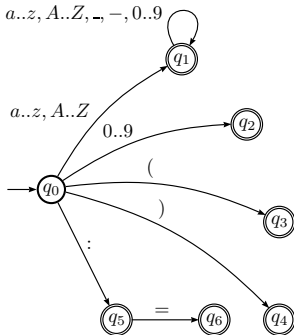
- Detección de **errores léxicos**
- **Guardar información** de los **tokens**, necesaria para el proceso de compilación.

Nota

El **analizador léxico** también se conoce como scanner, lexer o tokenizer.

3. Herramientas para implementar analizadores léxicos

- Autómatas finitos...



- Generadores de analizadores léxicos:

- Lex, Flex (<http://flex.sourceforge.net/>)
- JFlex (<http://jflex.de/>)
- Ragel (<http://www.complang.org/ragel/>)
- ANTLR (<http://www.antlr.org/>)
- ...

4. Implementación de analizadores léxicos en ANTLR

Definición de Tokens

Palabras reservadas

```
@tokens
{
    INT_TYPE      = "int"      ;
    REAL_TYPE     = "real"     ;
    BOOL_TYPE     = "bool"     ;
    STRING_TYPE   = "string"   ;
    ...
    BEGIN_RW      = "begin"    ;
    END_RW        = "end"      ;
    IF_RW         = "if"       ;
    ELSE_RW       = "else"     ;
    WHILE_RW      = "while"    ;
    ...
    TRUE_LITERAL  = "true"     ;
    FALSE_LITERAL = "false"    ;
    ...
}
```

4. Implementación de analizadores léxicos en ANTLR

Definición de Reglas

Letras

```
LETTER : 'a'..'z'  
       | 'A'..'Z'  
       ;
```

Dígitos

```
DIGIT : '0'..'9';
```

Identificadores

```
IDENT  
options {testLiterals=true;} // Comprobar palabras reservadas  
      : (LETTER|'_' ) (LETTER|DIGIT|'_' ) *  
      ;
```

4. Implementación de analizadores léxicos en ANTLR

Definición de Reglas

Símbolos de puntuación

```
SEMICOLON : ';' ;
COMMA     : ',' ;
DOT       : '.' ;
COLON    : ':' ;
...
```

Paréntesis

```
LEFT_PAREN  : '(' ;
RIGHT_PAREN : ')' ;
LEFT_BRACE  : '{' ;
RIGHT_BRACE : '}' ;
LEFT_BRACKET : '[' ;

RIGHT_BRACKET
//opción para mensajes de error
options { paraphrase="a right bracket (']')"; }
: ']' ;
;
```

4. Implementación de analizadores léxicos en ANTLR

Definición de Reglas

Operadores Matemáticos

```
PLUS   : '+' ;  
SUB    : '-' ;  
STAR   : '*' ;  
SLASH  : '/' ;  
...
```

Operadores Relacionales

```
EQUAL      : '=' ;  
NOT_EQUAL  : '!=' ;  
GR_EQUAL   : '>' ;  
LE_EQUAL   : '<' ;  
...
```


4. Implementación de analizadores léxicos en ANTLR

Definición de Reglas

Números

```

NUM_LITERAL : (( DIGIT )+ '.' ) =>
              ( DIGIT )+ '.' ( DIGIT )+ { $setType (REAL_LITERAL); }
              | ( DIGIT )+                { $setType (INT_LITERAL); }
              ;
  
```

Blancos

```

WS :    ((' ' | '\t' | '\f')
        | ( '\n' | '\r' ) { newline(); }
        ) { _ttype = Token.SKIP; }
        ;
  
```

5. Ejercicios

Implemente el analizador léxico del lenguaje MiLe++ (Micro Lenguaje)

Tokens

```
numeric  extends  if    and    true
string   object   else  not   false
class    method  for   print
main     return  or    read
```

Puntuación, paréntesis y operadores

```
. , ; { } [] ( ) + - * / < > != == <= >= =
```

Números e identificadores

```
number ::= (digit)+ ("." (digit)+)?
ident  ::= (letter|'_' ) (letter|digit|'_' )*
```