Universidad
Rey Juan Carlos

# Procesadores de Lenguajes

## [PRÁCTICA OBLIGATORIA]

ÁLVARO SIRVENT MARTÍN Y RUBÉN BARGUEÑO PRIETO

# Tabla de contenido

# Autores

**Nombre:** **Álvaro Sirvent Martín**

**Grado que se está cursando:** **Doble grado en ingeniería informática e ingeniería de computadores.**

**Nombre:** **Rubén Bargueño Prieto**

**Grado que se está cursando:** **Doble grado en ingeniería informática e ingeniería de computadores.**

# Descripción del Código

## Especificación léxica del lenguaje

```
ID        : [a-zA-Z_] [a-zA-Z_0-9]* ;

CONSTINT  : [0-9]+ | [+-] [0-9]+;

CONSTREAL
   :  [+-] [0-9]+ '.' [0-9]+                    // Punto fijo con signo
   |  [0-9]+ '.' [0-9]+                         // Punto fijo sin signo
   |  [+-] [0-9]+ [Ee] [+-] [0-9]+              // Exponencial con ambos signos
   |  [+-] [0-9]+ [Ee] [0-9]+                   // Exponencial con signo solo al inicio
   |  [0-9]+ [Ee] [+-] [0-9]+                   // Exponencial con signo solo en exponente
   |  [0-9]+ [Ee] [0-9]+                        // Exponencial sin ningún signo
   |  [+-] [0-9]+ '.' [0-9]+ [Ee] [+-] [0-9]+   // Mixto con ambos signos
   |  [+-] [0-9]+ '.' [0-9]+ [Ee] [0-9]+        // Mixto con signo solo al inicio
   |  [0-9]+ '.' [0-9]+ [Ee] [+-] [0-9]+        // Mixto con signo solo en exponente
   |  [0-9]+ '.' [0-9]+ [Ee] [0-9]+             // Mixto sin ningún signo
   ;

CONSTLIT  : '\'' (~['\r\n])* '\'' ;

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

COMMENT   : '//' ~[\r\n]* -> skip ;
```

# Gramática LL(1)

La gramática elaborada para esta práctica debía ser BNF a solicitud del enunciado y LL(1), para el correcto funcionamiento con la herramienta ANTLR. Este tipo de gramáticas deben cumplir las siguientes condiciones:

- No pueden presentar ambigüedad en las alternativas de sus producciones, es decir, los conjuntos FIRST deben ser disjuntos.
- Si hay producciones vacías, para mantener lo establecido en la norma previa, los conjuntos FIRST Y CAB deben de ser disjuntos también.
- No puede presentar recursividad por la izquierda, ni directa ni indirecta.

A la hora de construir la gramática se tuvieron en cuenta estas condiciones, de modo que no ha sido necesario hacer ninguna transformación intermedia para obtenerla. Se ha escogido la regla "for_sent" para explicar detalladamente el método de método de análisis que se ha seguido para obtener los conjuntos DIR de la gramática PascalLike y justificar que es LL(1).

```
for_sent returns [String result]
    : 'for' ID ':=' e1=exp inc e2=exp 'do'
    {
        $result = "for(" + $ID.text + "=" + $e1.text + "; " ;
        if($inc.result == "to") $result += $ID.text + "<=" + $e2.text + "; " + $ID.text + "++)";
        else if($inc.result == "downto") $result += $ID.text + ">=" + $e2.text + "; " + $ID.text + "--)";
        $result += " {\n";
    }
    bloque
    {$result += $bloque.main + "\n" + getTab() + "}";}
    ;
```

La regla comienza con el token "for" y esta es la única regla de "sent" que empieza por este token. Por lo que no hay ambigüedad a nivel del símbolo inicial en "sent".

Cada paso consume exactamente un token que es único en su contexto:

- "for": obligatorio
- ID: obligatoriamente un identificador
- ":=": operador de asignación
- exp: una expresión aritmética
- inc: token "to" o "downto"
- Otro exp: otra expresión
- "do": palabra clave literal
- bloque: también LL(1)

La regla "for_sent" solo tiene una producción, así que no puede haber conflictos de los conjuntos de cabecera ni siguiente, por lo tanto la única producción de "for_sent" comienza con un terminal exclusivo ("for"), garantizando determinismo en la elección de la regla, todos los símbolos internos son obligatorios y no existen producciones de lambda, por lo que no aparecen solapamientos entre

los conjuntos cabecera y siguientes, las subreglas (“exp”, “bloque”) están diseñadas para ser LL(1) de forma recursiva, sin introducir conflictos.

Se analizará también la regla “exp” asociado a la regla anterior “for_sent”.

```
exp returns [String result]
    : term exp_tail {$result = $term.result + $exp_tail.result;}
    ;

exp_tail returns [String result]
    : '+' term tail=exp_tail    {$result = " + " + $term.result + " " + $tail.result;}
    | '-' term tail=exp_tail    {$result = " - " + $term.result + " " + $tail.result;}
    | /*vacío*/                 {$result = "";}
    ;

term returns [String result]
    : factor term_tail {$result = $factor.result + $term_tail.result;}
    ;

term_tail returns [String result]
    : '*' factor term_tail      {$result = " * " + $factor.result + " " + $term_tail.result;}
    | 'div' factor term_tail    {$result = " / " + $factor.result + " " + $term_tail.result;}
    | 'mod' factor term_tail    {$result = " % " + $factor.result + " " + $term_tail.result;}
    | /*vacio */                {$result = "";}
    ;
```

**Conjuntos cabecera:**

CAB(exp) = {CONSTREAL, CONSTINT, CONSTLIT, '(', ID}

CAB(exp_tail) = { '+', '-', lambda }

CAB(term) = {CONSTREAL, CONSTINT, CONSTLIT, '(', ID}

CAB(term_tail) = { '*', 'div', 'mod', lambda }

**Conjuntos siguientes:**

SIG(exp) = {',', ';', 'end', 'then', 'do', 'until', 'to', 'downto', ')'}

SIG(exp_tail) = SIG(exp)

SIG(term) = { '+', ' -' } U SIG(exp)

SIG(term_tail) = SIG(term)

**Conjuntos directores:**

DIR(exp → term exp_tail) = {CONSTREAL, CONSTINT, CONSTLIT, '(', ID}

DIR(exp_tail → '+' term exp_tail) = {'+'}

DIR(exp_tail → '-' term exp_tail) = {'-'}

DIR(exp_tail → lambda) = SIG(exp_tail) = {',', ';', 'end', 'then', 'do', 'until', 'to', 'downto', ')'}

DIR(term → factor term_tail) = {CONSTREAL, CONSTINT, CONSTLIT, '(', ID}

DIR(term_tail → '*' factor term_tail) = {'*'}

DIR(term_tail → 'div' factor term_tail) = {'div'}

DIR(term_tail → 'mod' factor term_tail) = {'mod'}

DIR(term_tail → lambda) = SIG(term_tail) = {',', ';', 'end', '+', '-', 'then', 'do', 'until', 'to', 'downto', ')'}

Tras este análisis podemos confirmar que la regla no presenta recursividad por la izquierda y sus conjuntos FIRST son disjuntos.

En la siguiente tabla se exponen las reglas de la gramática junto con sus conjuntos FIRST, CAB Y DIR.

| No Terminal | Producción | DIR |
|---|---|---|
| **programa** | 'program' ID ';' bloque '.' | {'program'} |
| | 'unit' ID ';' declist '.' | {'unit'} |
| **bloque** | declist 'begin' sentlist 'end' | {'const', 'var', 'procedure', 'function', 'begin'} |
| **declist** | declist_item_list | {'const', 'var', 'procedure', 'function', '.', 'begin'} |
| **declist_item_list** | declist_item declist_item_list | {'const', 'var', 'procedure', 'function'} |
| | λ | {'.', 'begin'} |
| **declist_item** | constdecl | {'const'} |
| | vardecl | {'var'} |
| | procdecl | {'procedure'} |
| | funcdecl | {'function'} |
| **constdecl** | 'const' ctelist | {'const'} |
| **ctelist** | ID '=' simovalue ';' ctelist_tail | {ID} |
| **ctelist_tail** | ctelist | {ID} |
| | λ | {'const', 'var', 'procedure', 'function', '.', 'begin'} |
| **simovalue** | CONSTREAL | {CONSTREAL} |
| | CONSTINT | {CONSTINT} |
| | CONSTLIT | {CONSTLIT} |
| **vardecl** | 'var' defvarlist | {'var'} |
| **defvarlist** | varlist ':' tbas ';' defvarlist_tail | {ID} |
| **defvarlist_tail** | defvarlist | {ID} |
| | λ | {'const', 'var', 'procedure', 'function', '.', 'begin'} |

| No Terminal | Producción | DIR |
|---|---|---|
| **varlist** | ID varlist_tail | {ID} |
| **varlist_tail** | ',' varlist | {','} |
| | λ | {':'} |
| **tbas** | 'INTEGER' | {'INTEGER'} |
| | 'REAL' | {'REAL'} |
| **procdecl** | 'procedure' ID formal_paramlist_opt ';' bloque ';' | {'procedure'} |
| **funcdecl** | 'function' ID formal_paramlist_opt ':' tbas ';' bloque ';' | {'function'} |
| **formal_paramlist_opt** | formal_paramlist | {'('} |
| | λ | {';', ':'} |
| **formal_paramlist** | '(' formal_param formal_param_tail ')' | {'('} |
| **formal_param_tail** | ';' formal_param formal_param_tail | {';'} |
| | λ | {')'} |
| **formal_param** | varlist ':' tbas | {ID} |
| **sentlist** | sent sent_tail | {ID, 'if', 'while', 'repeat', 'for'} |
| **sent_tail** | ';' sent_tail_aux | {';'} |
| | λ | {'end'} |
| **sent_tail_aux** | sent sent_tail | { ID, 'if', 'while', 'repeat', 'for' } |
| | λ | { 'end' } |

| No Terminal | Producción | DIR |
|---|---|---|
| **sent** | ID sent_id_tail | {ID} |
| | if_sent | {'if'} |
| | while_sent | {'while'} |
| | repeat_sent | {'repeat'} |
| | for_sent | {'for'} |
| **sent_id_tail** | asig | {':='} |
| | proc_call | {'(', ';', 'end'} |
| **asig** | ':=' exp | {':='} |
| **proc_call** | subparamlist_opt | { '(', ';', 'end' } |
| **subparamlist_opt** | subparamlist | {'('} |
| | λ | {';', 'end', '*', 'div', 'mod', ',', ')', '+', '-', 'then', 'do', 'until', 'to', 'downto'} |
| **subparamlist** | '(' explist_opt ')' | {'('} |
| **explist_opt** | explist | {CONSTREAL, CONSTINT, CONSTLIT, '(', ID} |
| | λ | {')'} |
| **explist** | exp explist_tail | {CONSTREAL, CONSTINT, CONSTLIT, '(', ID} |
| **explist_tail** | ',' explist | {','} |
| | λ | {')'} |
| **exp** | term exp_tail | {CONSTREAL, CONSTINT, CONSTLIT, '(', ID} |
| **exp_tail** | '+' term exp_tail | {'+'} |
| | '-' term exp_tail | {'-'} |
| | λ | {',', ';', 'end', 'then', 'do', 'until', 'to', 'downto', ')'} |
| **term** | factor term_tail | {CONSTREAL, CONSTINT, CONSTLIT, '(', ID} |

| No Terminal | Producción | DIR |
|---|---|---|
| **term_tail** | '*' factor term_tail | {'*'} |
| | 'div' factor term_tail | {'div'} |
| | 'mod' factor term_tail | {'mod'} |
| | λ | {',', ';', 'end', '+', '-', 'then', 'do', 'until', 'to', 'downto', ')'} |
| **factor** | simovalue | {CONSTINT, CONSTREAL, CONSTLIT} |
| | '(' exp ')' | {'('} |
| | ID subparamlist_opt | {ID} |
| **exp_no_paren** | term_no_paren exp_tail | {CONSTREAL, CONSTINT, CONSTLIT, ID} |
| **term_no_paren** | factor_no_paren term_tail | {CONSTREAL, CONSTINT, CONSTLIT, ID} |
| **factor_no_paren** | simovalue | {CONSTREAL, CONSTINT, CONSTLIT} |
| | ID subparamlist_opt | {ID} |
| **expcond** | and_term expcond_tail | {'not', '(', CONSTREAL, CONSTINT, CONSTLIT, ID} |
| **expcond_tail** | 'or' and_term expcond_tail | {'or'} |
| | λ | {'then', 'do', ')', 'until', ';', 'end'} |
| **and_term** | not_term and_term_tail | {'not', '(', CONSTREAL, CONSTINT, CONSTLIT, ID} |
| **and_term_tail** | 'and' not_term and_term_tail | {'and'} |
| | λ | {'or', 'then', 'do', ')', 'until', ';', 'end'} |
| **not_term** | 'not' not_term | {'not'} |
| | factorcond | {'('}, {CONSTREAL, CONSTINT, CONSTLIT, ID} |
| **factorcond** | '(' expcond ')' | {'('} |
| | exp_no_paren opcomp exp | {CONSTREAL, CONSTINT, CONSTLIT, ID} |

| No Terminal | Producción | DIR |
|---|---|---|
| **opcomp** | '=' | {'='} |
| | '<>' | {'<>'} |
| | '<' | {'<'} |
| | '>' | {'>'} |
| | '<=' | {'<='} |
| | '>=' | {'>='} |
| **if_sent** | 'if' expcond 'then' bloque else_block_opt | {'if'} |
| **else_block_opt** | 'else' else_block | {'else'} |
| | λ | {';', 'end'} |
| **else_block** | bloque | {'const', 'var', 'procedure', 'function', 'begin'} |
| | if_sent | {'if'} |
| **while_sent** | 'while' expcond 'do' bloque | {'while'} |
| **repeat_sent** | 'repeat' bloque 'until' expcond | {'repeat'} |
| **for_sent** | 'for' ID ':=' exp inc exp 'do' bloque | {'for'} |
| **inc** | 'to' | {'to'} |
| | 'downto' | {'downto'} |

Tras evaluar todos los conjuntos DIR de la tabla y confirmar que para cada regla son disjuntos, podemos determinar que la gramática elaborada es LL(1).

# Casos de prueba

## Casos correctos.

**Input 1:** debido a la longitud del código se ha optado por no incluir las capturas de pantalla de este. No obstante, este código buscaba comprobar el funcionamiento de la mayor cantidad de funciones y reglas posibles, evaluando:

- Constantes y variables, tanto globales como locales.
- "Procedure".
- Declaración de funciones.
- Asignación de variables.
- Sentencias condicionales ("if","else-if"…).
- Instrucciones cíclicas ("while", "repeat"…).
- Llamada a funciones.
- Cierre de código.

**Input 3:**

```
1 program ejemploWhile;
2 var
3   contador: INTEGER;
4 begin
5   contador := 0;
6   while contador < 5 do
7     begin
8       write('Contador: ', contador);
9       contador := contador + 1;
10    end;
11 end.
```

**Input 7:**

```
1 program ejemploCondiciones;
2 var
3   a, b, c: INTEGER;
4 begin
5   a := 10;
6   b := 20;
7   c := 30;
8   if (a < b) and (b < c) then
9     begin
10      write('a < b < c');
11    end
12  else if not (a = b) then
13    begin
14      write('a no es igual a b');
15    end;
16 end.
```

**Input 8:**

```
unit miLibreria;
const
  PI = 3.1416;
var
  global: INTEGER;
procedure saludar;
begin
  write('Hola desde la librería');
end;
.
```

## Casos erróneos.

**Input 2:** "if" incorrecto. En la línea 7 falta un "then".

```
program ejemploIf;
var
  x, y: INTEGER;
begin
  x := 10;
  y := 20;
  if x > y
    begin
      write('x es mayor que y');
    end
  else
    begin
      write('y es mayor o igual que x');
    end;
end.
```

**Input 4:** "repeat" incorrecto. En la línea 4 debería aparecer "begin"

```
program ejemploRepeat;
var
  contador: INTEGER;

  contador := 0;
  repeat
    begin
      write('Contador: ', contador);
      contador := contador + 1;
    end
  until contador >= 5;
end.
```

**Input 5:** "for-to" incorrecto. En la última línea le falta el "." al "end".

```
1 program ejemploForTo;
2 var
3   i: INTEGER;
4 begin
5   for i := 1 to 5 do
6     begin
7       write('Iteración: ', i);
8     end;
9 end
```

**Input 6:** "for-downto" incorrecto. Se ha añadido la línea 7, que debería estar vacía.

```
1 program ejemploForDownto;
2 var
3   i: INTEGER;
4 begin
5   for i := 5 downto 1 do
6     begin
7       error
8       write('Iteración: ', i);
9     end;
10 end.
```