PL - Procesadores de Lenguajes - extracto.pdf

Asignatura Completa

3° Procesadores de Lenguajes

Grado en Ingeniería Informática - Tecnologías Informáticas

Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla

TRINITY COLLEGE LONDON
Registered Exam Centre 67874

We prepare for Cambridge English Qualifications

INGLÉS O FRANCÉS

IDIOMAS PARA TODOS LOS GUSTOS

4 MESES → *4 horas a la semana*

¡SIMULACROS REALES DE EXAMEN!

DESDE 69€/MES

A2

C1 B1

B2

CLASES ONLINE Y PRESENCIALES

Plazas Limitadas · Material didáctico incluido

MÉNDEZ NÚÑEZ
Academia de Enseñanza

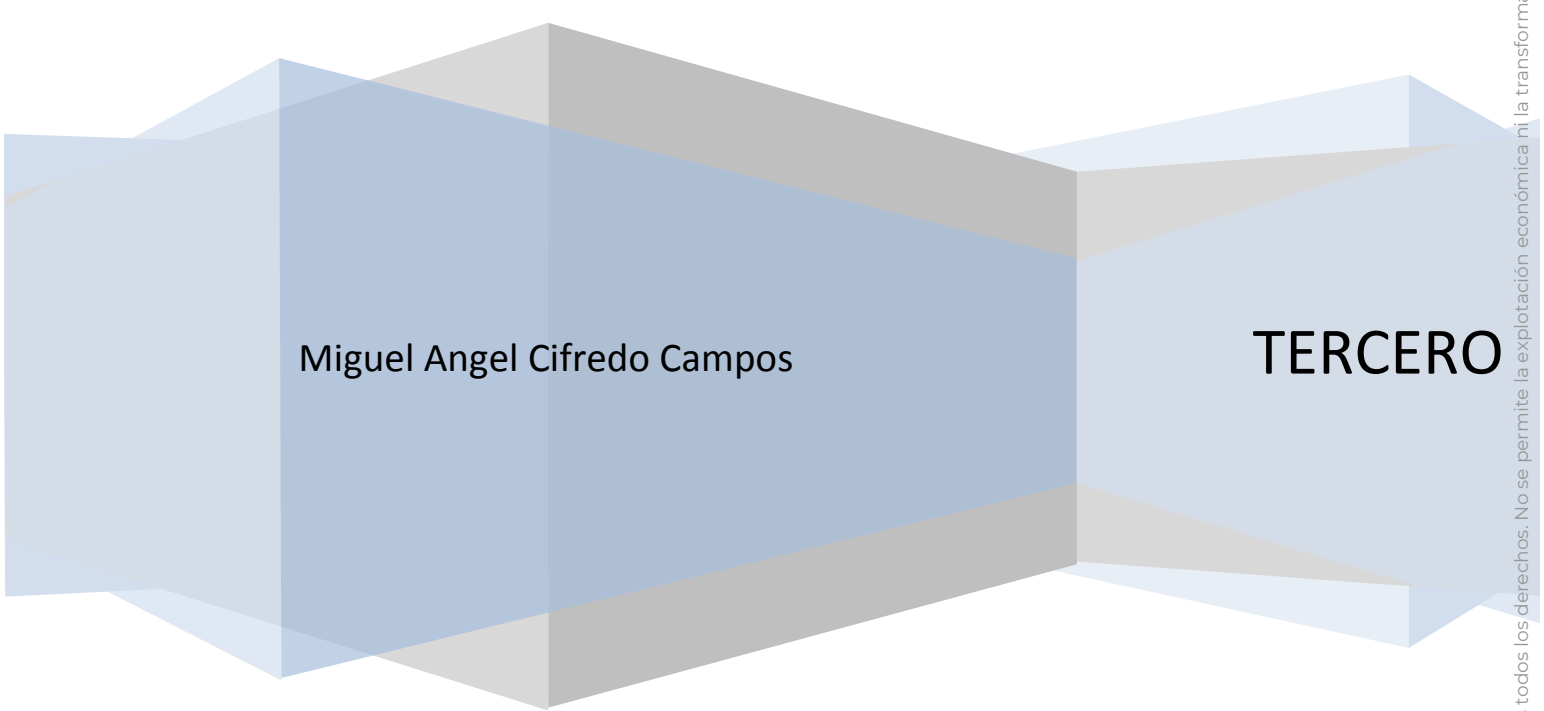
954 225 225
www.academiamn.com

C/Méndez Núñez 1, 2ª planta. 41001 Sevilla

Escuela Técnica Superior de Ingeniería Informática

Procesadores de Lenguajes

Departamento de Lenguajes y Sistemas Informáticos



Miguel Angel Cifredo Campos

TERCERO

--- REVERSO DE PORTADA ---

1 Introducción.

1.1 Objetivo.

Todo lenguaje posee sintaxis y semántica. La sintaxis se refiere a la *forma* de las sentencias del lenguaje. La semántica se refiere al *significado* de tales sentencias. La especificación de lenguaje se realizará de manera informal y basada en ejemplos típicos.

El diseño del procesador está basado en un patrón arquitectónico denominado *tubería-filtro*. Sobre una *tubería* se desplazan distintas representaciones de datos o de programas que son procesadas por *filtros* especializados.

Hay múltiples tecnologías para construir procesadores de lenguajes. En el presente curso utilizaremos una tecnología llamada **ANTLR** (acrónimo de *ANother Tool for Language Recognition*).

1.2 Especificación del lenguaje.

La especificación del lenguaje a procesar estará basada en definiciones informales y ejemplos típicos.

Ejemplo 1. Especificación de un lenguaje de programación.

Definición informal de un lenguaje hipotético de programación al que llamaremos LF: Se trata de un lenguaje de programación secuencial con variables enteras y dos tipos de instrucciones: Definición de variables con expresión entera (DEF) y, Evaluación de variables (EVAL). No se acepta el uso de variables sin declarar. La declaración de variables asocia valor 0 por defecto. Cuando se define una variable se le asocia una expresión entera "sin evaluar". Esta expresión se evalúa justo al ejecutar la instrucción EVAL. Toda variable definida sobre sí misma se le asocia un valor indefinido.

A continuación se muestra un programa de ejemplo para ver una sintaxis del lenguaje LF.

```
VARIABLES x, y, z, a, b;
INSTRUCCIONES
      a DEF -1;
      b DEF (a+1);
      EVAL b;
      a DEF 2;
      EVAL b;
      b DEF b + 1;
      z DEF 2*y;
      EVAL z;
      EVAL b;
```

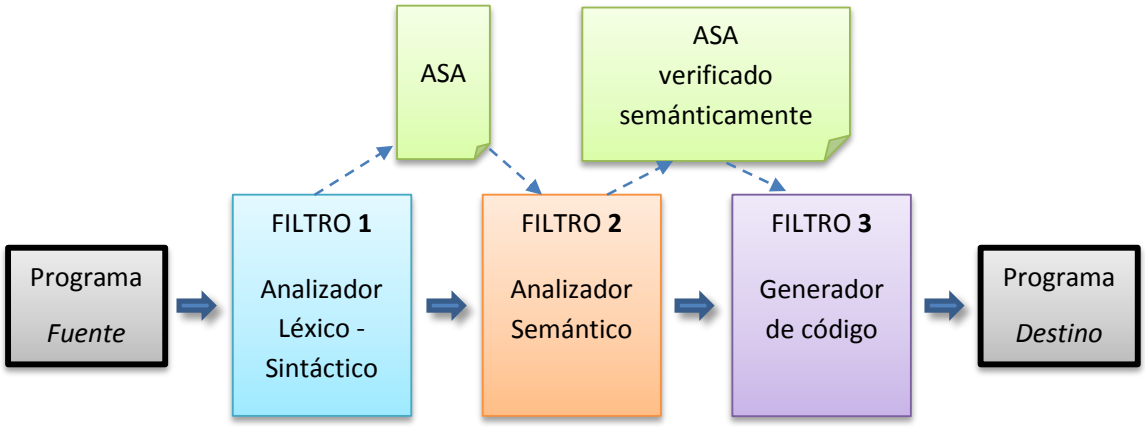
Para precisar la semántica del lenguaje LF, se aporta la ejecución del programa anterior:

```
      b = 0             // primera instrucción EVAL
      b = 3             // segunda instrucción EVAL
      z = 0             // tercera instrucción EVAL
      b = indefinido    // cuarta  instrucción EVAL
```

1.3 Diseño del procesador.

El diseño del procesador se basará en el patrón arquitectónico denominado *tubería-filtro*. Sobre la tubería se desplazan distintas representaciones de datos o programas que son procesadas por filtros especializados. La naturaleza y número de filtros dependerá del problema a resolver.

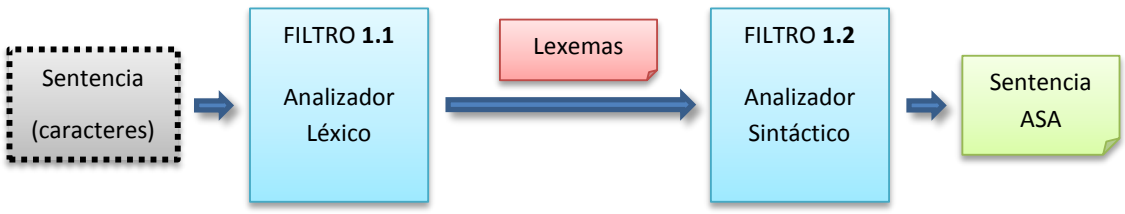
El siguiente gráfico muestra la arquitectura de un compilador con tres filtros: Analizador léxico-sintáctico, Analizador semántico y Generador de código. Partiendo de un programa fuente, en forma de secuencia de caracteres, se generará un árbol de sintaxis abstracta (asa) (Filtro 1). A continuación se comprueba, con un analizador semántico (Filtro 2), si el árbol de sintaxis abstracta (asa) cumple condiciones semánticas suficientes para asignar semántica al programa fuente. Si esto es así, el árbol de sintaxis abstracta (asa), verificado semánticamente, se puede usar como entrada a un generador de código (Filtro 3). Este último filtro traducirá el programa original a un programa en el lenguaje destino.



1.4 Filtro 1: Análisis Léxico-Sintáctico.

Su objetivo es decidir si la descripción textual de entrada tiene la forma adecuada.

El analizador léxico-sintáctico es en realidad un doble filtro. El primer filtro es el *analizador léxico* que toma la sentencia entrada en forma de secuencia de caracteres y produce una salida en forma de secuencia de lexemas. El segundo filtro es el *analizador sintáctico* que toma la secuencia de lexemas producido por el analizador léxico y produce una versión de la entrada en forma de árbol de sintaxis abstracta (asa).



Lexema: Unidad textual (palabra o signo de puntuación) utilizados en el lenguaje a procesar. Los lexemas son unidades mínimas desde las que se construyen estructuras sintácticas. El ejemplo 2 muestra la secuencia de lexemas en la que se transforma el programa mostrado en el ejemplo 1. Entre paréntesis se muestra la palabra o signo de puntuación asociado al lexema. Se indicarán en mayúsculas.

Ejemplo 2. Programa del lenguaje LF como secuencia de lexemas.

```
VARIABLES(VARIABLES), VAR(x), COMA(,), VAR(y), COMA(,), VAR(z), COMA(,), VAR(a), COMA(,), VAR(b),
PyC(;), INSTRUCCIONES(INSTRUCCIONES), VAR(a), DEF(DEF), MENOS(-), NUMERO(1), PyC (;), VAR(b),
lexema:10(DEF), PA((), VAR(a), MAS(+), NUMERO (1), PC()), PyC (;), EVAL(EVAL), VAR(b), PyC(;),
VAR(a), DEF(DEF), NUMERO (2), PyC (;), EVAL(EVAL), VAR(b), PyC (;), VAR(b), DEF(DEF), VAR(b),
MAS(+), NUMERO(1), PyC (;), VAR(z), DEF(DEF), NUMERO (2), POR(*), VAR(y), PyC (;), EVAL(EVAL),
VAR(z), PyC (;), EVAL(EVAL), VAR(b), PyC (;)
```

Ejemplo 3. Analizador Léxico.

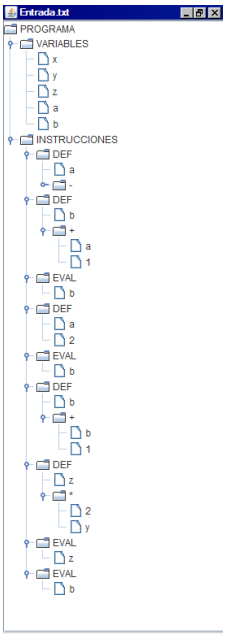
El siguiente ejemplo muestra el código ANTLR del *analizador léxico* que genera la secuencia de lexemas mostrada en el ejemplo 2.

```
class Analex extends Lexer;

options{
        importVocab    = Anasint;
}

tokens{
        VARIABLES     = "VARIABLES";
        INSTRUCCIONES = "INSTRUCCIONES";
        EVAL          = "EVAL";
        DEF           = "DEF";
}

protected NL     : "\r\n"  {newline();} ;
protected DIGITO : '0'..'9';
protected LETRA  : 'a'..'z'|'A'..'Z';

BTF    : (' '|'\t'|NL)     {$setType(Token.SKIP);} ;
NUMERO : (DIGITO)+ ;
VAR    : (LETRA)+ ;
PA     : '(' ;
PC     : ')' ;
PyC    : ';' ;
COMA   : ',' ;
MAS    : '+' ;
MENOS  : '-' ;
POR    : '*' ;
```

Árbol de Sintaxis Abstracta (asa): Es un árbol construido desde lexemas de acuerdo a las reglas sintácticas del lenguaje.

Ejemplo 4. Programa LF como árbol de sintaxis abstracta (asa).

Se muestra el árbol de sintaxis abstracta (asa) construido desde la secuencia de lexemas mostrado en el ejemplo 2 para el programa mostrado en ejemplo 1.



Ejemplo 5. Analizador sintáctico.

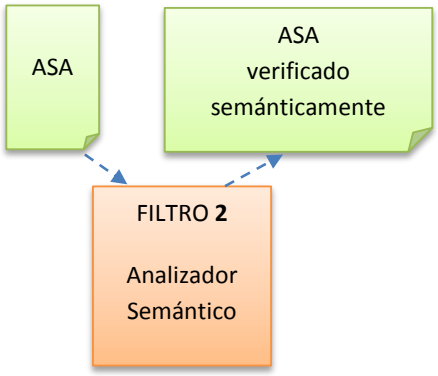
El ejemplo 5 muestra el código ANTLR del *analizador sintáctico* que genera el árbol de sintaxis abstracta (asa) mostrado en el ejemplo 4.

```
class Anasint extends Parser;
options{
        buildAST = true;
}
tokens{
        PROGRAMA;
}
programa      : variables instrucciones EOF!
                {#programa = #(#[PROGRAMA, "PROGRAMA"], ##);}
              ;
variables     : VARIABLES^ idents PyC!
              ;
idents        : (VAR COMA) => VAR COMA! idents
              | VAR
              |
              ;
instrucciones : INSTRUCCIONES^ (definicion|evaluacion)*
              ;
definicion    : VAR DEF^ expr PyC!
              ;
evaluacion    : EVAL^ idents PyC!
              ;
expr          : (expr1 MAS)   => expr1 MAS^ expr
              | (expr1 MENOS) => expr1 MENOS^ expr
              | expr1
              ;
expr1         : (expr2 POR)   => expr2 POR^ expr1
              | expr2
              ;
expr2         : NUMERO
              | MENOS^ expr
              | VAR
              | PA! expr PC!
              ;
```

Tanto el *analizador léxico* como el *analizador sintáctico* pueden detectar errores en sus procesamientos respectivos. Los errores detectados por el analizador léxico se denominan *errores léxicos* y los detectados por el analizador sintáctico se denominan *errores sintácticos*.

1.5 Filtro 2: Análisis Semántico.

¿Podemos asociarle significado a un programa que ha superado el filtro léxico-sintáctico? Generalmente no. El filtro léxico-semántico no es capaz de procesar condiciones sintácticas dependientes del contexto tales como el uso de variables no declaradas o condiciones semánticas tales como la corrección del tipo asociado a las expresiones. El objetivo del filtro semántico es comprobar el cumplimiento de tales condiciones.



Ejemplo 6. Analizador semántico.

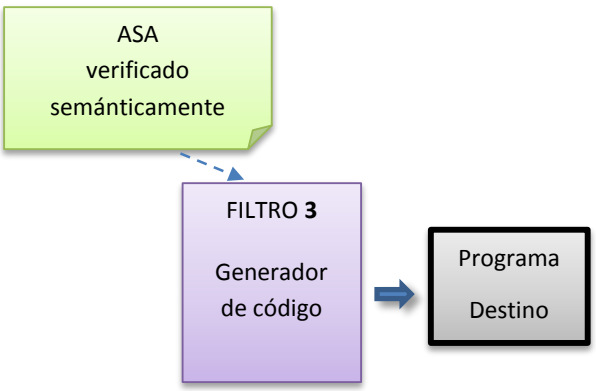
El ejemplo 6 muestra el código ANTLR del *analizador semántico* que verifica el asa mostrado en el ejemplo 4.

```
header{
      import java.util.*;
      import antlr.*;
}
class Anasint4 extends TreeParser;

options {
      importVocab = Anasint;
}
{
      Set<String> variables = new HashSet<String>();
      ASTFactory factory    = new ASTFactory();
      public void almacenar(String var){
            variables.add(var);
      }
      public boolean ocurrencia(String var, AST expr){
            switch(expr.getType()){
                  case NUMERO:      return false;
                  case VAR:         return expr.getText().equals(var);
                  case MAS:
                  case POR:         return ocurrencia(var, expr.getFirstChild())
                                        || ocurrencia(var, expr.getFirstChild().getNextSibling());
                  case MENOS: if (expr.getFirstChild().getNextSibling()==null)
                                    return ocurrencia(var, expr.getFirstChild());
                              else
                                    return ocurrencia(var, expr.getFirstChild())
                                        || ocurrencia(var, expr.getFirstChild().getNextSibling());
            }
            return false;
      }
}
programa      : #(PROGRAMA variables instrucciones )
              ;
variables     : #(VARIABLES (v:VAR {almacenar(v.getText());} )*)
              ;
instrucciones : #(INSTRUCCIONES (definicion|evaluacion)*)
              ;
definicion    : #(DEF v:VAR e:expr)
              { if (!variables.contains(v.getText()))
                   System.out.println("Variable no declarada: " + #v.getText());
                if (ocurrencia(#v.getText(), #e))
                   System.out.println("Variable indefinida: " + #v.getText());
              } ;
evaluacion    : #(EVAL v:VAR)
              ;
expr          : #(MAS expr expr)
              | #(POR expr expr)
              | NUMERO
              | VAR
              | (#(MENOS expr expr)) => #(MENOS expr expr)
              | #(MENOS expr)
              ;
```

Los errores detectados por el analizador semántico se denominan *errores semánticos*.

1.6 Filtro 3: Generador de Código.

El objetivo de la generación de código es traducir el programa a otro lenguaje (normalmente ejecutable). El programa que supera el filtro semántico es un programa al que se le puede asociar significado. Por tanto, este programa se puede ejecutar, bien interpretándolo directamente o bien a través de una traducción a otro lenguaje. El generador de código no produce errores, es una función total.



En nuestro ejemplo, la generación de código consiste en convertir el programa original en una versión Java de éste.

Ejemplo 7. Generador de código.

```
header{
      import java.util.*;
      import antlr.*;
      import java.io.*;
}

class Anasint3 extends TreeParser;

options
{
      importVocab = Anasint;
}


{
      FileWriter fichero;

      private void open_file(){
            try{
                  fichero = new FileWriter("_Programa.java");
            }catch(IOException e)
            { System.out.println("open_file (exception): " + e.toString()); }
      }


      private void close_file(){
            try{
                  fichero.close();
            }catch(IOException e)
               { System.out.println("close_file (exception): " + e.toString()); }
      }


      int espacios = 0;
      private void gencode_espacios(){
            try{
                  for (int i = 1; i <= espacios; i++)
                        fichero.write(" ");
            }catch(IOException e)
                 { System.out.println("gencode_espacios (exception): " + e.toString()); }
      }
```

```
ASTFactory             factory    = new ASTFactory();
Hashtable<String, AST> vars       = new Hashtable<String, AST>();
Hashtable<String, AST> vars2      = new Hashtable<String, AST>();
List<String>           evals_pend = new LinkedList<String>();
Integer                num_evals  = new Integer(0);

public void declarar_variable(String var){
      AST nodo = new CommonAST();
      nodo.setType(NUMERO);
      nodo.setText("0");
      vars.put(var,nodo);
}

public void definir_variable(String var, AST expr){
      vars.put(var,expr);
}

public void evaluar_variable(String var){
      Integer i;
      String aux = new String(var);
      num_evals++;
      Enumeration<String> v = vars.keys();
      while (v.hasMoreElements()){
            String key = v.nextElement();
            if (cierre(var).contains(key)){
                  aux = key + num_evals.toString();
                  vars2.put(aux,sustitucion_expr(vars.get(key)));
            }
      }
      try{
            v = vars2.keys();
            while (v.hasMoreElements()){
                  String key = v.nextElement();
                  gencode_espacios();
                  fichero.write("private static int " + key + "(){\n");
                  espacios++;
                  gencode_espacios();
                  fichero.write("return " + gencode_exp(vars2.get(key)) + ";\n");
                  espacios--;
                  gencode_espacios();
                  fichero.write("}\n");
            }
      }catch(IOException e){}
      vars2.clear();
      evals_pend.add(var+num_evals.toString());
}

public Set<String> cierre(String var){
      Set<String> resultado = new HashSet<String>();
      int size_ant = 0;
      int size_act = 0;
      resultado.add(var);
      size_act = resultado.size();
      while (size_ant != size_act){
            resultado.addAll(calcular_variables(vars.get(var)));
            size_ant = size_act;
            size_act = resultado.size();
      }
      return resultado;
}

public Set<String> calcular_variables(AST expr){
      Set<String> aux = new HashSet<String>();
      switch(expr.getType()){
            case NUMERO: break;
            case VAR:          aux.add(expr.getText()); break;
            case MAS:
            case POR:          aux.addAll(calcular_variables(expr.getFirstChild()));
                               aux.addAll(calcular_variables(expr.getFirstChild().getNextSibling()));
                         break;
            case MENOS:        aux.addAll(calcular_variables(expr.getFirstChild()));
                         if (expr.getFirstChild().getNextSibling() != null)
                               aux.addAll(calcular_variables(expr.getFirstChild().getNextSibling()));
                         break;
            default:           aux = null;
      }
      return aux;
}
```

```
public AST sustitucion_expr(AST expr){
      AST nodo;
      String aux;
      switch(expr.getType()){
            case NUMERO: nodo = factory.dupTree(expr);
                         break;
            case VAR   : aux = expr.getText();
                         nodo = new CommonAST();
                         aux = aux + num_evals.toString();
                         nodo.setType(VAR);
                         nodo.setText(aux);
                         break;
            case MAS   : nodo = new CommonAST();
                         nodo.setType(MAS);
                         nodo.setText("+");
                         nodo.setFirstChild(sustitucion_expr(expr.getFirstChild()));
                         nodo.getFirstChild().setNextSibling(
                               sustitucion_expr(expr.getFirstChild().getNextSibling())
                         );
                         break;
            case POR   : nodo = new CommonAST();
                         nodo.setType(POR);
                         nodo.setText("*");
                         nodo.setFirstChild(sustitucion_expr(expr.getFirstChild()));
                         nodo.getFirstChild().setNextSibling(
                               sustitucion_expr(expr.getFirstChild().getNextSibling())
                         );
                         break;
            case MENOS : nodo = new CommonAST();
                         nodo.setType(MENOS);
                         nodo.setText("-");
                         nodo.setFirstChild(sustitucion_expr(expr.getFirstChild()));
                         if (expr.getFirstChild().getNextSibling()!=null)
                            nodo.getFirstChild().setNextSibling(
                                 sustitucion_expr(expr.getFirstChild().getNextSibling())
                            );
                         break;
            default:     nodo = null;
      }
      return nodo;
}

private void gencode_begin_class(){
      try{
            gencode_espacios();
            fichero.write("import java.io.*;\n");
            gencode_espacios();
            fichero.write("public class _Programa" + "\n");
            gencode_espacios();
            fichero.write("{\n");
            espacios++;
      }catch(IOException e){}
}

private void gencode_main(){
      try{
            gencode_espacios();
            fichero.write("public static void main(String[] args) {\n");
            espacios++;
            Iterator<String> it = evals_pend.listIterator();
            while (it.hasNext()){
                  gencode_espacios();
                  fichero.write("System.out.println(" + it.next() + "());\n");
            }
            espacios--;
            gencode_espacios();
            fichero.write("}\n");
      }catch(IOException e){}
}

private void gencode_end_class(){
      try{
            espacios--;
            gencode_espacios();
            fichero.write("}");
      }catch(IOException e){}
}
```

```
      public String gencode_exp(AST expr){
            switch(expr.getType()){
                  case NUMERO: return expr.getText();
                  case VAR   : return expr.getText() + "()";
                  case MAS   : return
gencode_exp(expr.getFirstChild()) + "+" + gencode_exp(expr.getFirstChild().getNextSibling());
                  case POR   : return
gencode_exp(expr.getFirstChild()) + "*" + gencode_exp(expr.getFirstChild().getNextSibling());
                  case MENOS : if (expr.getFirstChild().getNextSibling() != null)
                                     return
gencode_exp(expr.getFirstChild()) + "-" + gencode_exp(expr.getFirstChild().getNextSibling());
                               else
                                     return "-" + gencode_exp(expr.getFirstChild());
                  default    : return null;
               }
      }

}

programa      : {open_file(); gencode_begin_class();}
                #(PROGRAMA variables instrucciones)
                {gencode_main(); gencode_end_class(); close_file();}
              ;
variables     : #(VARIABLES (v:VAR {declarar_variable(v.getText());} )*)
              ;
instrucciones : #(INSTRUCCIONES (definicion|evaluacion)*)
              ;
definicion    : #(DEF v:VAR e:expr) {definir_variable(v.getText(), e);}
              ;
evaluacion    : #(EVAL v:VAR)       {evaluar_variable(v.getText());}
              ;
expr          : #(MAS expr expr)
              | #(POR expr expr)
              | NUMERO
              | VAR
              | (#(MENOS expr expr)) => #(MENOS expr expr)
              | #(MENOS expr)
              ;
```

Ejemplo 8. Código Java generado para el programa del lenguaje LF.

A continuación se presenta el código Java generado para el programa del lenguaje LF del ejemplo 1.

```
import java.io.*;

public class _Programa {
     private static int a1() {
        return -1;
     }
     private static int b1() {
        return a1() + 1;
     }
     private static int a2() {
        return 2;
     }
     private static int b2() {
        return a2() + 1;
     }
     private static int y3() {
        return 0;
     }
     private static int z3() {
        return 2 * y3();
     }
     private static int b4() {
        return b4() + 1;
     }
     public static void main(String[] args) {
        System.out.println(b1());
        System.out.println(b2());
        System.out.println(z3());
        System.out.println(b4());
     }
}
```

Miguel Ángel Cifredo Campos

macifredo@gmail.com

https://es.linkedin.com/in/miguelangelcifredo