# Tema 1.5. Un lenguaje mínimo y su procesador: *Traducción*


**Profesor**
Federico Peinado

**Elaboración del material**
José Luis Sierra
Federico Peinado

**Ingeniería en Informática**
**Facultad de Informática – Universidad Complutense de Madrid**
**Curso 2009-2010**

# Especificación de la tradución

- Hasta ahora nos hemos centrado en definir formalmente un lenguaje, muy útil de cara a poder *analizarlo*… pero también hay que especificar cómo será su *síntesis*
  - En el caso de un traductor, cómo será su **traducción** a otro lenguaje (= lenguaje objeto)

- Para esto también usaremos una **gramática de atributos** y también nos apoyaremos en la **tabla de símbolos**
  - La tabla de símbolos tendrá que ser *ampliada* para que contenga información adicional, como la dirección de memoria asignada a cada variable del programa

# Ampliación del ejemplo:Especificación para hacer la traducción

- Para poder traducir del lenguaje fuente del ejemplo al lenguaje objeto de la máquina P es necesario especificar **cómo se realiza dicha traducción entre instrucciones** (cómo se genera el código P a partir del árbol sintáctico del código fuente y su tabla de símbolos)

- También, en el caso de este lenguaje de ejemplo, es necesario especificar todo lo relativo a las **direcciones de memoria** asociadas a los identificadores
  - Ampliar la definición de la TS para que cada identificador tenga asociada una dirección
  - Ampliar la operación de inserción en la TS para poder ir añadiendo dichas direcciones al insertar identificadores
  - Ampliar la gramática de atributos que especifica la construcción de la TS para tener en cuenta las direcciones

# Registro de propiedades de los identificadores

- En lenguajes de programación más completos hay mucha otra información que asociar con los identificadores (clasificación como *constante* o *variable*, *tipo*, *ámbito*, etc.)

- Para generalizar, hacemos que todos los identificadores se asocien en la TS con un **registro de sus propiedades**
  - Se representará con el tipo:
    **Propiedades**
  - Cada instancia del registro se construye con la tupla:
    ***<propiedad1:valor1, …, propiedadN:valorN>***
  - Se accederá a una propiedad de un identificador así:
    **TS[*lexema _identificador*].*propiedadX***

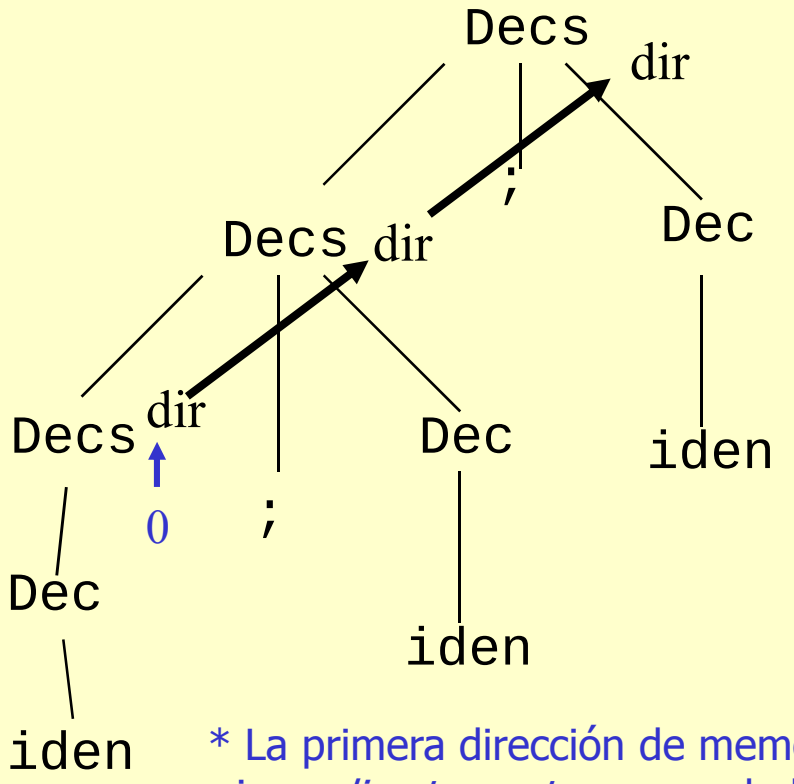
# Ampliación del ejemplo:Especificación para hacer la traducción

- **Operaciones** de la tabla de símbolos
  - Descripción informal de la semántica de las operaciones

**creaTS():TS**
El resultado es una TS vacía
**añadeID(ts:TS, id:String, ps: Propiedades):TS**
El resultado es la TS resultante de añadir *id* y sus propiedades *ps* a *ts*
**existeID(ts: TS, id:String):Boolean**
El resultado es *true* si *id* aparece en *ts*, *false* en caso contrario
**ts:TS[id:String]:Propiedades**
El resultado es el registro de propiedades de *id* (<> si *id* no está en la TS)

# Ampliación del ejemplo:Especificación para hacer la traducción

- **Construcción** de la tabla de símbolos
  - Ej. La dirección como atributo sintetizado en la sección de declaración de variables




* La primera dirección de memoria (la 0) en realidad se asigna *directamente* en una de las ecuaciones semánticas

# Ampliación del ejemplo:Especificación para hacer la traducción

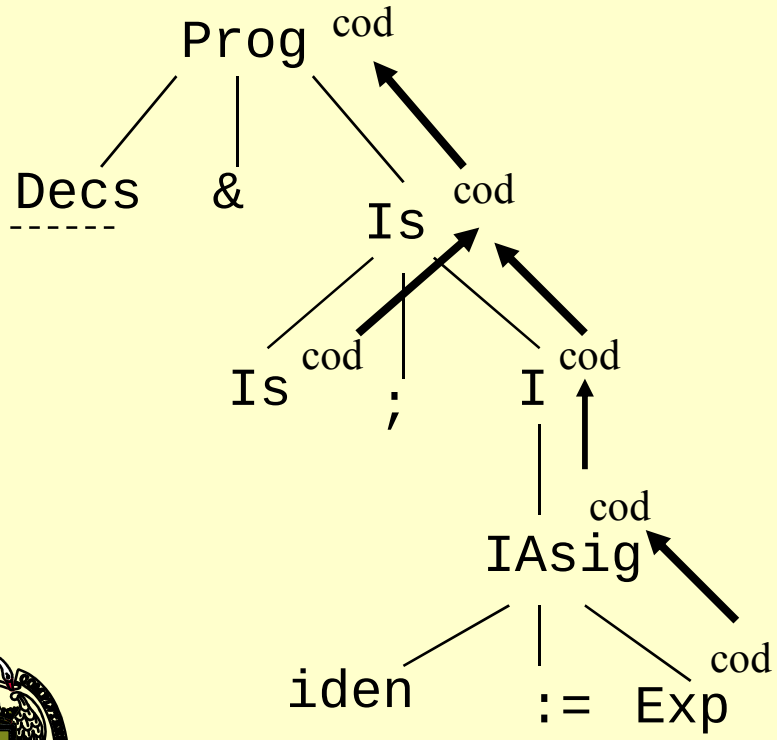
- **Construcción** de la tabla de símbolos
  - Ej. La dirección como atributo sintetizado en la sección de declaración de variables

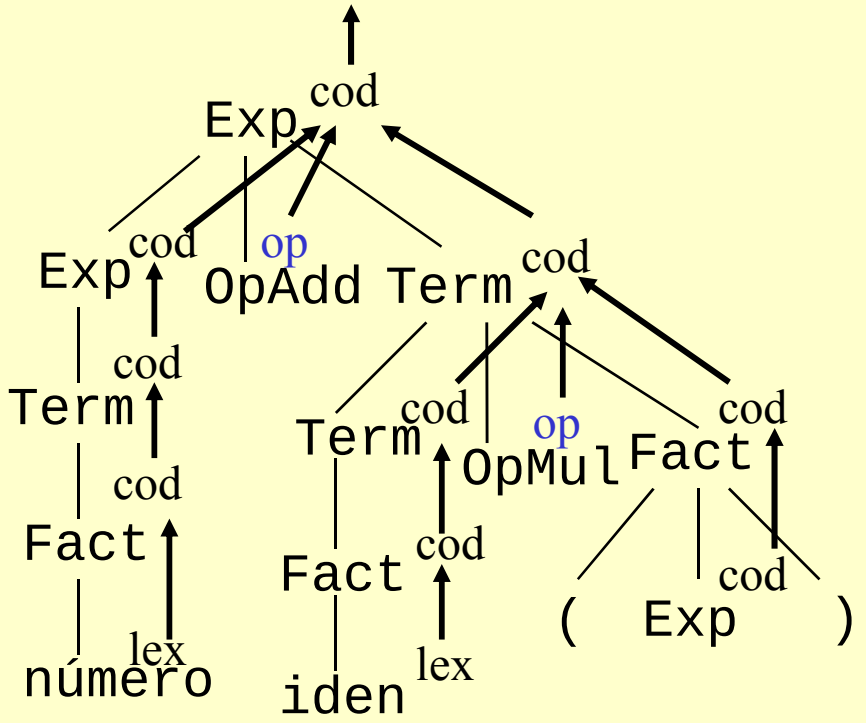
```
Decs ::= Dec
  Decs.dir = 0
  Decs.ts = añadeID(creaTS(), Dec.id, <dir:0>)
Decs ::= Decs ; Dec
  Decs0.ts = añadeID(Decs1.ts, Dec.id, <dir:Decs1.dir +1>)
  Decs0.dir = Decs1.dir+1
```

* El **dir** que está en negrita no es el dir *atributo semántico* de la gramática, sino una *propiedad* del registro de propiedades de los identificadores

# Ampliación del ejemplo:Especificación para hacer la traducción

- **Generación de código**, en principio, como un atributo sintetizado llamado *cod*




* **op** es un atributo similar a lex: es un enumerado que nos dice cual ha sido exactamente el operador utilizado

# Ampliación del ejemplo:Especificación para hacer la traducción

- **Generación de código**, en principio, como un atributo sintetizado llamado *cod*

```
Prog ::=  Decs & Is
   Prog.cod = Is.cod || stop
Is ::= I
   Is.cod = I.cod
Is::= Is ; I
   Is_o.cod = Is_1.cod || I.cod
I ::= IAsig
   I.cod = IAsig.cod
IAsig ::= iden := Exp
   Iasig.cod = Exp.cod ||
               despila-dir(IAsig.ts_h[iden.lex].dir)
Exp ::= Exp OpAd Term
   Exp_o.cod = Exp_1.cod || Term.cod || OpAd.op
Exp  ::= Term
   Exp.cod = Term.cod
...
```

* Recordatorio: **||** significa concatenar el fragmento de código P de la izquierda por delante del fragmento de código P de la derecha

# Ampliación del ejemplo:Especificación para hacer la traducción

- **Generación de código**, en principio, como un atributo sintetizado llamado *cod (continuación…)*

```
Term ::= Term OpMul Fact
  Termo.cod = Term1.cod || Fact.cod || OpMul.op
Term  ::= Fact
  Term.cod = Fact.cod
Fact ::= iden
  Fact.cod = apila-dir(Fact.tsh[iden.lex].dir)
Fact ::= num
   Fact.cod = apila(valorDe(num.lex)))
Fact ::= ( Exp )
   Fact.cod = Exp.cod
OpAd ::= +
    OpAd.op = suma
OpAd ::= -
   OpAd.op = resta
OpMul ::= *
   OpMul.op = multiplica
OpMul ::= /
   OpMul.op = divide
```

# Críticas, dudas, sugerencias...

Federico Peinado
www.federicopeinado.es