4.pdf

Anónimo

Procesadores de Lenguajes

3º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

PROCESADORES DE LENGUAJES

10 de enero de 2018

Observaciones: 1. Las calificaciones se publicarán hacia el 22 de enero.
2. La revisión será hacia el 24 de enero.
3. En la web se avisarán las fechas exactas.
4. La duración de este examen es de 40 minutos.

3. Sea un lenguaje con las siguientes características:

- Tiene los tipos int, long, byte, void, float, boolean y char. Esto quiere decir que existen reglas para los tipos (T → int | long | byte | void | float | boolean | char).
- Tiene expresiones (E → id | E op-ar E | E op-rel E | E op-lóg E | (E) | id [E] | cte_entera | cte_real | ...).
- Exige declaración previa de variables
- Las variables y los vectores pueden ser de cualquier tipo del lenguaje, salvo void.
- Los tipos int, long y byte son variantes del tipo "entero" y son compatibles entre sí. Para el resto de tipos, el lenguaje no tiene conversión automática entre ellos.
- Al asignar valores a un vector completo (id={L}), para cada elemento del vector tiene que haber uno y solo un valor.
- La variable índice (id) de la sentencia from se declara en la propia sentencia y ha de ser de alguno de los tres tipos "entero".
- La sentencia from funciona de la siguiente manera: se realiza la asignación id=E; si la expresión del when es cierta, se ejecuta el cuerpo del from y se vuelve al inicio; si es falsa, se sale del from.

Se pide diseñar el Analizador Semántico mediante un Esquema de Traducción únicamente para las siguientes reglas de la gramática (usando el espacio proporcionado para cada regla):

```
D → { zona_decl := true } T id                          /* declaración de una variable */
{ if T.tipo ≠ vacio
  then InsertarTipoTS(id.entrada, T.tipo)
       InsertarDesplTS(id.entrada, despl)
       despl:= despl + T.tamaño
  else error()  /* las variables no pueden ser de tipo void */
zona_decl := false }
```

```
D → { zona_decl := true }                    /* declaración de un vector de "cte_entera" elementos */
    T id [ 1 .. cte_entera ]
{ if T.tipo ≠ vacio
  then InsertarTipoTS(id.entrada, array(1..cte_entera.val, T.tipo))
       InsertarDesplTS(id.entrada, despl)
       despl:= despl + T.tamaño * cte_entera.val
  else error()  /* los elementos de un vector no pueden ser de tipo void */
zona_decl := false }
```

```
S → id = E                                              /* asignación de un valor a una variable */
{ id.tipo:= BuscaTipoTS(id.entrada)
  S.tipo:= if (id.tipo = E.tipo ≠ tipo_error) OR
              (id.tipo ∈ {int, long, byte} AND E.tipo ∈ {int, long, byte})
           then tipo_ok
           else tipo_error  /* tipos incompatibles en la asignación */ }
```

```
S → id [ cte_entera ] = E                               /* asignación de un valor a un elemento del vector */
{ S.tipo:= if BuscaTipoTS(id.entrada) = array(1..n, t)
           then if E.tipo = t OR
                   (t ∈ {int, long, byte} AND E.tipo ∈ {int, long, byte})
                then if 0 < cte_entera.val ≤ n
                     then tipo_ok
                     else tipo_error  /* el vector id tiene solo "n" elementos */
                else tipo_error /* la expresion no es del tipo de los elementos del vector */
           else tipo_error  /* la variable id no es un vector */}
```

```
S → id = {L}                                            /* asignación de valores a un vector completo */
{ S.tipo:= if BuscaTipoTS(id.entrada) = array (1..n, t)
           then if L.tipo = t OR
                   (t ∈ {int, long, byte} AND L.tipo ∈ {int, long, byte})
                then if L.núm_elem = n
                     then tipo_ok
                     else tipo_error
                          /* Tiene que haber uno y solo un valor para cada elemento del vector */
                else tipo_error
```

```
{ if T.tipo ≠ vacio
  then InsertarTipoTS(id.entrada, T.tipo)
       InsertarDesplTS(id.entrada, despl)
       despl:= despl + T.tamaño
  else error()  /* las variables no pueden ser de tipo void */
zona_decl := false }
```

```
D → { zona_decl := true }                    /* declaración de un vector de "cte_entera" elementos */
    Tid [ 1 .. cte_entera ]
{ if T.tipo ≠ vacio
  then InsertarTipoTS(id.entrada, array(1..cte_entera.val, T.tipo))
       InsertarDesplTS(id.entrada, despl)
       despl:= despl + T.tamaño * cte_entera.val
  else error()  /* los elementos de un vector no pueden ser de tipo void */
zona_decl := false }
```

```
S → id = E                                   /* asignación de un valor a una variable */
{ id.tipo:= BuscaTipoTS(id.entrada)
  S.tipo:= if (id.tipo = E.tipo ≠ tipo_error) OR
              (id.tipo ∈ {int, long, byte} AND E.tipo ∈ {int, long, byte})
           then tipo_ok
           else tipo_error  /* tipos incompatibles en la asignación */ }
```

```
S → id [ cte_entera ] = E                    /* asignación de un valor a un elemento del vector */
{ S.tipo:= if BuscaTipoTS(id.entrada) = array(1..n, t)
           then if E.tipo = t OR
                   (t ∈ {int, long, byte} AND E.tipo ∈ {int, long, byte})
                then if 0 < cte_entera.val ≤ n
                     then tipo_ok
                     else tipo_error  /* el vector id tiene solo "n" elementos */
                else tipo_error /* la expresion no es del tipo de los elementos del vector */
           else tipo_error  /* la variable id no es un vector */}
```

```
S → id = { L }                               /* asignación de valores a un vector completo */
{ S.tipo:= if BuscaTipoTS(id.entrada) = array (1..n, t)
           then if L.tipo = t OR
                   (t ∈ {int, long, byte} AND L.tipo ∈ {int, long, byte})
                then if L.núm_elem = n
                     then tipo_ok
                     else tipo_error
                          /* Tiene que haber uno y solo un valor para cada elemento del vector */
                else tipo_error
                          /* los valores asignados no son del tipo de los elementos del vector */
           else tipo_error  /* La variable id no es un vector */ }
```

```
L → E { L.tipo:= E.tipo                      /* una expresión */
        L.núm_elem:= 1 }
```

```
L → E, L1                                    /* una lista de expresiones */
{ L.tipo:= if E.tipo = L1.tipo OR
             (E.tipo ∈ {int, long, byte} AND L1.tipo ∈ {int, long, byte})
           then E.tipo
           else tipo_error  /* todos los elementos han de ser del mismo tipo (o "entero") */
  L.núm_elem:= 1 + L1.núm_elem }
```

```
S → from { zona_decl := true }  Tid          /* sentencia repetitiva */
{ if T.tipo ∈ {int, long, byte}
  then InsertarTipoTS(id.entrada, T.tipo)
       InsertarDesplTS(id.entrada, despl)
       despl:= despl + T.tamaño
  else error()  /* tipo erróneo en el índice del from. Debe ser de algún tipo "entero" */ }
zona_decl := false }
   = E1 when E2 { S1 }
{ S.tipo:= if T.tipo ∈ {int, long, byte}
           then if E1.tipo ∈ {int, long, byte}
                then if E2.tipo = lógico
                     then S1.tipo
                     else tipo_error  /* la expresión del when debe ser de tipo lógico */
                else tipo_error /* el valor asignado a id no es de uno de los tipos "entero" */
           else tipo_error  /* declaración errónea en la sentencia from */ }
```

PROCESADORES DE LENGUAJES

SOLUCIÓN ANÁLISIS SINTÁCTICO

10 de enero de 2018