MANUAL DE USO DEL PROCESADOR DE BOREAL

Introducción

El presente manual pretende aportar los fundamentos para ayudar en la comprensión, modificación y personalización del Analizador del Procesador de lenguaje Boreal de cara a desarrollar la práctica de Traductores de Lenguajes.

En el Procesador de Boreal se ha utilizado la librería del gestor de Tablas de Símbolos: TS_Gestor. Por tanto, deberá consultarse su correspondiente manual para saber cómo utilizar las Tablas de Símbolos.

Tras una introducción general del Procesador, se comentan algunos detalles sobre las acciones semánticas del Analizador Semántico que pueden ser de utilidad para la construcción del Generador de Código Intermedio.

Procesador de Boreal

El Procesador de Boreal incluye un **Analizador Léxico** completo para todo el lenguaje. Cada grupo de prácticas tendrá solamente algunas de las opciones del lenguaje, pero se recomienda no modificar el Analizador Léxico (aunque reconozca más elementos de los necesarios para el grupo de prácticas). Este Analizador Léxico funciona como un servicio que proporciona al Analizador Sintáctico el siguiente *token* disponible en el fichero de entrada, además de introducir en la Tabla de Símbolos correspondiente todos los identificadores que aparezcan en el programa fuente.

El Procesador también incluye un **Analizador Sintáctico Ascendente** LR(1), en concreto, SLR. Igualmente, el Analizador Sintáctico es para el lenguaje Boreal completo. De nuevo, se recomienda no modificar el Analizador Sintáctico (aunque reconozca más declaraciones, sentencias y expresiones de las necesarias para el grupo de prácticas). El anexo 1 contiene la gramática utilizada para construir el Analizador Sintáctico.

Integrado con el Analizador Sintáctico se dispone de un **Analizador Semántico** correspondiente al lenguaje completo. Igual que en los casos anteriores, se recomienda no eliminar nada, aunque no se corresponda con las opciones de la práctica asignada al grupo. La gramática del Analizador Sintáctico y el Analizador Semántico se han diseñado de forma que no sea necesario el uso de atributos heredados en la Traducción Dirigida por la Sintaxis, por lo que todos los atributos serán sintetizados. El anexo 2 muestra el Esquema de Traducción utilizado para desarrollar el Analizador Semántico.

Juntamente con los tres módulos de Análisis, se dispone de una **Tabla de Símbolos** completa, que permite almacenar toda la información necesaria sobre todos los identificadores de Boreal (además de las palabras reservadas). Los analizadores completarán estas Tablas de Símbolos (Global y Locales) con la información obtenida por el Analizador Léxico y por el Analizador Semántico. Por tanto, aunque la información es completa hasta ese punto, podría ser necesario incluir nueva información necesaria para la fase de síntesis del Compilador.

Finalmente, se dispone de un sencillo **Gestor de Errores** que, en función del error enviado por el resto de los módulos, informarán al usuario de los posibles errores detectados durante el análisis del programa fuente.

Para la construcción del Compilador, deberá añadirse el **Generador de Código Intermedio** y el **Generador de Código Objeto**. El primero deberá integrarse junto al Analizador Semántico existente, para que funcione con el Analizador Léxico y el Analizador Sintáctico. Por ello, en los siguientes apartados de este manual, se darán algunas indicaciones sobre cómo incorporar al Analizador Semántico el Esquema de Traducción del Generador de Código Intermedio. El segundo se añadirá como un nuevo módulo que reciba los cuartetos del Generador de Código Intermedio para traducirlos directamente a ensamblador.

Analizador Semántico

Estructura de la Clase ASem

Variables de la Clase

La clase que representa el Analizador Semántico contiene las siguientes variables de clase:

```
public class ASem {
    public static boolean tsGlobal;                          // Indica si existe TS global
    private static boolean zonaDeclaracion;   // Indica si se está en zona de declaraciones
    private static Integer despGlobal, despLocal;       // Desplazamientos de las variables
    private static Integer numEtiq;                        // Número de etiquetas generadas
    private static final Map<Integer, Supplier<Atributos>> ruleMap = new HashMap<>();
                                                            // Información sobre las reglas
       …
}
```

- ***tsGlobal***: Define el estado de la tabla de símbolos, almacenando *true* si la tabla global es la activa, o *false* en caso contrario.
- ***zonaDeclaracion***: Define el estado de la zona de declaración, almacenando *true* si la zona de declaración está activa, o *false* en caso contrario.
- ***despGlobal***, ***despLocal***: Acumula los desplazamientos en la tabla de símbolos global y local, respectivamente.
- ***numEtiq***: Contador de las etiquetas asignadas.
- ***ruleMap***: Aporta información de las reglas.

Inicialización

En la primera llamada a la clase, se efectúa la inicialización de las variables globales, la declaración de los atributos de la tabla de símbolos y la inserción de las acciones semánticas en el mapa de reglas.

```
static {
                                    // Se configuran los atributos de la Tabla de Símbolos:
   Procesador.gestorTS.createAtributo("desplazamiento",
       DescripcionAtributo.DIR, TipoDatoAtributo.ENTERO);
   Procesador.gestorTS.createAtributo("etiqueta",
       DescripcionAtributo.ETIQUETA, TipoDatoAtributo.CADENA);
   Procesador.gestorTS.createAtributo("PasoParametros",
       DescripcionAtributo.MODO_PARAM, TipoDatoAtributo.LISTA);
   Procesador.gestorTS.createAtributo("tipoParametros",
       DescripcionAtributo.TIPO_PARAM, TipoDatoAtributo.LISTA);
   Procesador.gestorTS.createAtributo("tipoRetorno",
       DescripcionAtributo.TIPO_RET, TipoDatoAtributo.CADENA);
   Procesador.gestorTS.createAtributo("numParametro",
       DescripcionAtributo.NUM_PARAM, TipoDatoAtributo.ENTERO);
   Procesador.gestorTS.createAtributo("modoParametro",
       DescripcionAtributo.PARAM, TipoDatoAtributo.ENTERO);
                                                // Se inicializan las variables de la clase:
   zonaDeclaracion = false;
   tsGlobal = true;
   numEtiq = 2;
                                        // Se asignan las acciones semánticas a cada regla:
   ruleMap.put(1, ASem::acc1);
   ruleMap.put(2, ASem::acc2);
   ruleMap.put(3, ASem::acc3);
   …
   ruleMap.put(100, ASem::acc100);
}
```

Diseño de una Acción Semántica

Atributos

Una acción semántica se representa por medio de una función que retorna una clase contenedora de atributos, denominada `Atributos`, localizada en el fichero ASem.java.

La clase `Atributos` contiene las variables y métodos para manejar los atributos de cada símbolo gramatical, como posición (`pos`), tipo (`tipo`), cantidad de sentencias *exit* (`exit`), tamaño del tipo (`ancho`), etc.

El acceso y modificación de cada atributo se realiza mediante las correspondientes funciones `get` y `set` de la clase `Atributos`.

```
class Atributos {
   private Integer pos;                          // Atributo posición en la Tabla de Símbolos
   private String  tipo;                                                  // Atributo tipo
   private Integer exit;             // Atributo para contar el número de instrucciones exit
   private String  ret;                                    // Atributo para el tipo de retorno
   private Integer ancho;                       // Atributo para el tamaño de un tipo de datos
   private Integer longs;                    // Atributo para calcular la longitud de una lista
   private String  referencia;      // Atributo que indica si un parámetro es por referencia
   private String  etiqueta;                              // Atributo para guardar una etiqueta
   private Integer program_count;  // Atributo para contar cuántos programas principales hay

   public Atributos() {        // Constructor que inicializa todas las variables de la clase
       this.pos = null;
       this.tipo = null;
       this.exit = null;
       this.ret = null;
       this.ancho = null;
       this.longs = null;
       this.referencia = null;
       this.etiqueta = null;
       this.program_count = null;
   }
                                       // Funciones get y set para las variables de la clase:
   public void    setPos(Integer pos) { this.pos = pos; }
   public Integer getPos() { return pos; }

   public void   setTipo(String tipo) { this.tipo = tipo; }
   public String getTipo() { return tipo; }

   public void    setExit(Integer exit) { this.exit = exit; }
   public Integer getExit() { return exit; }

   public void   setRet(String ret) { this.ret = ret; }
   public String getRet() { return ret; }

   public void    setAncho(Integer ancho) { this.ancho = ancho; }
   public Integer getAncho() { return ancho; }

   public void    setLong(Integer longs) { this.longs = longs; }
   public Integer getLongs() { return longs; }

   public void   setReferencia(String ref) { this.referencia = ref; }
   public String getReferencia() { return referencia; }

   public void   setEtiqueta(String et) { this.etiqueta = et; }
   public String getEtiqueta() { return etiqueta;}

   public void setProgramCount(Integer i) { this.program_count = i; }
   public int  getProgramCount() { return this.program_count; }
   …
}
```

Si se desean añadir más atributos, se debe crear la nueva variable en la clase con el tipo deseado, inicializarlo en el constructor y definir las respectivas funciones `set` y `get` de dicho nuevo atributo.

Por ejemplo, si fuera necesario crear un nuevo atributo (ficticio) para contabilizar la cantidad de bloques *begin-end*, se podría hacer de la siguiente manera:

```
class Atributos {
   …
   private Integer begin_cont;                                          // Nuevo atributo

   public Atributos() {
   …
        this.begin_cont = null;        // Inicialización del nuevo atributo en el constructor
   }
   …
                                                                  // get y set para el nuevo atributo
   public void    setBegin_Cont(Integer begin_cont) { this.begin_cont = begin_cont; }
   public Integer getBegin_Cont() { return this.begin_cont; }
}
```

Acción Semántica

Por lo general, las acciones semánticas se desarrollan en tres etapas:

1. Obtención de los atributos de la pila de atributos:
   ```
   Regla reg = MT_ASINT.getRegla(1);
   Atributos[] atb = ASin.pilaSem.toArray(new Atributos[reg.numElementos * 2]);
   ```
   - ***reg*** almacena la información de las reglas.
   - A partir de la lista ***atb***, se pueden obtener los atributos de cada componente de la regla:
     - Si la regla 1 tiene la forma 'P → D a K E', siendo 'a' un símbolo terminal y 'D', 'K' y 'E' símbolos no terminales, los atributos de 'D', 'K' y 'E' se almacenan en los índices 7, 3 y 1 respectivamente en la lista ***atb***.
2. Desarrollo de la acción semántica.
3. Devolución de los atributos del antecedente de la regla ('P', en este caso).

Modificación de una Acción Semántica

Si se desea modificar una Acción Semántica, por ejemplo, para añadir las acciones semánticas del Generador de Código Intermedio, se debe modificar directamente la función correspondiente a la acción semántica que se quiere ampliar.

Por ejemplo, si se quisiera realizar una modificación (ficticia) sobre la regla 26 (Bloque → begin C end), se parte de la acción semántica actual:

Bloque → begin C end { Bloque.tipo:= C.tipo
Bloque.exit:= C.exit
Bloque.tipoRet:= C.tipoRet }

El código correspondiente a la acción semántica de la regla 26 es:

```
private static Atributos acc26() {                          // Acción semántica de la regla 26
   Regla reg= MT_ASINT.getRegla(26);              // Se obtiene la información de la regla 26
   Atributos[] atb= ASin.pilaSem.toArray(new Atributos[reg.numElementos * 2]);
                                        // Se crean los atributos correspondientes a la regla
   Atributos res= new Atributos();         // Se crean los atributos del antecedente (Bloque)
   res.setTipo(atb[3].getTipo());                                  // Bloque.tipo:= C.tipo
   res.setExit(atb[3].getExit());                                  // Bloque.exit:= C.exit
   res.setRet(atb[3].getRet());                                 //Bloque.tipoRet:= C.tipoRet
   return res;                          // Se devuelven los atributos del antecedente (Bloque)
}
```

Ahora, si se supone que la nueva acción semántica que se quiere realizar para dicha regla es:

Bloque → begin C end { Bloque.tipo:= C.tipo
Bloque.exit:= C.exit
Bloque.tipoRet:= C.tipoRet
Bloque.BeginCont:= C.BeginCont + 1 }

El nuevo código de la acción semántica de la regla 26 sería:

```
private static Atributos acc26() {
   Regla reg= MT_ASINT.getRegla(26);
   Atributos[] atb= ASin.pilaSem.toArray(new Atributos[reg.numElementos * 2]);
   Atributos res= new Atributos();
   res.setTipo(atb[3].getTipo());
   res.setExit(atb[3].getExit());
   res.setRet(atb[3].getRet());
   res.setBegin_Cont(atb[3].getBegin_Cont() + 1);        // Bloque.BeginCont:= C.BeginCont+1
   res.setRet(ret);
   return res;
}
```

Creación de una Nueva Acción Semántica

Si se desea crear una nueva Acción Semántica en una regla que carece de acciones semánticas, solamente será necesario añadir una nueva entrada, si no existe ya, en el `HashMap` de reglas dentro de la clase `ASem` y añadir la implementación de la nueva función de la acción semántica correspondiente a dicha regla, estando dicha función definida tal como se ha indicado en los apartados anteriores.

Tipos de Datos

El Analizador Semántico maneja una serie de tipos asociados a los distintos símbolos gramaticales. Para ello, se utilizan cadenas con los nombres de los tipos escritos en minúsculas de la siguiente manera: “cadena”, “entero”, “lógico”, “función”, “procedimiento”, “tipo_ok” y “tipo_error”; para el tipo vacío se usa una cadena vacía “”.

Gestor de Errores

El Gestor de Errores permite informar al usuario los errores detectados durante la compilación. Básicamente, se utilizan dos formatos para enviar la información al usuario:

- **Flexible**: Se emplea la función `writeError` de la clase `GestorError` para imprimir un mensaje de error puntual.
  `GestorError.writeError("Exit fuera de bucle detectado en Procedure");`
- **Predefinido**: Se crea una instancia en la clase `Accion`, se añade un valor en el `enum Acciones` y una entrada en un nuevo `case` con el mensaje de error en la función `printError()` de la clase `GestorError`. La llamada se realiza usando la función `setError`, a la que se le puede enviar información adicional como segundo parámetro. Esta forma es la recomendada si el mensaje de error se puede producir en distintas situaciones.
  `GestorError.setError(Acciones.eSem5_case_otherwise_error, "");`

Anexo 1: Gramática del Analizador Sintáctico

```
//// Conjunto de símbolos terminales
Terminales = { write writeln read return var if else then boolean integer string loop end mod not cadena
      true false max min while repeat until exit when for id to do begin case of entero otherwise
      program procedure function in or xor and = <> > >= < <= := * ** / ( ) + - : ; , eof }
```

```
//// Conjunto de símbolos no terminales
NoTerminales = { P M1 R PP PPid Pid M2 PR PRidA PF PFidAT D DD T A AA Bloque C B EE THEN M3 FOR EXP N O
      VALOR S LL L Q V W Y E F G H I J K Z X }

//// Axioma
Axioma = P

//// Lista de producciones
Producciones = {
       P -> M1 D R
       M1 -> lambda
       R -> PP R
       R -> PF R
       R -> PR R
       R -> lambda
       PP -> program PPid ; D M2 Bloque ;
       PPid -> Pid
       Pid -> id
       M2 -> lambda
       PR -> procedure PRidA ; D M2 Bloque ;
       PRidA -> Pid A
       PF -> function PFidAT ; D M2 Bloque ;
       PFidAT -> Pid A : T
       D -> var id : T ; DD
       D -> lambda
       DD -> id : T ; DD
       DD -> lambda
       T -> boolean
       T -> integer
       T -> string
       A -> ( X id : T AA )
       A -> lambda
       AA -> ; X id : T AA
       AA -> lambda
       Bloque -> begin C end
       C -> B C
       C -> lambda
       B -> if EE then S
       EE -> E
       B -> S
       B -> if EE then Bloque ;
       B -> if THEN else Bloque ;
       THEN -> EE then Bloque ;
       B -> while M3 EE do Bloque ;
       M3 -> lambda
       B -> repeat M3 C until E ;
       B -> loop M3 C end ;
       B -> FOR do Bloque ;
       FOR -> for id := E to E
       B -> case EXP of N O end ;
       EXP -> E
       N -> N VALOR : Bloque ;
       VALOR -> entero
       N -> lambda
       O -> otherwise : M3 Bloque ;
       O -> lambda
       S -> write LL ;
       S -> writeln LL ;
       S -> read ( V ) ;
       S -> id := E ;
       S -> id LL ;
       S -> return Y ;
       S -> exit when E ;
       LL -> ( L )
       LL -> lambda
       L -> E Q
       Q -> , E Q
       Q -> lambda
       V -> id W
```

```
        W -> , id W
        W -> lambda
        Y -> E
        Y -> lambda
        E -> E or F
        E -> E xor F
        E -> F
        F -> F and G
        F -> G
        G -> G = H
        G -> G <> H
        G -> G > H
        G -> G >= H
        G -> G < H
        G -> G <= H
        G -> H
        H -> H + I
        H -> H - I
        H -> I
        I -> I * J
        I -> I / J
        I -> I mod J
        I -> J
        J -> J ** K
        J -> K
        K -> not K
        K -> + K
        K -> - K
        K -> Z
        Z -> entero
        Z -> cadena
        Z -> true
        Z -> false
        Z -> id LL
        Z -> ( E )
        Z -> Z in ( L )
        Z -> max ( L )
        Z -> min ( L )
        X -> var
        X -> lambda
}
```

Anexo 2: Esquema de Traducción del Analizador Semántico

1. **P → M1 D R** { If (R.program ≠ 1) Then Error (“Debe haber 1 y solo 1 Program”)
   DestruirTS (TSG)
   }

2. **M1 → λ** { TSG:= CrearTS ()
   TSActual:= TSG
   despl_global:= 0
   zona_decl:= true
   }

3. **R → PP R_1** { R.program:= 1 + R_1.program }
4. **R → PF R_1** { R.program:= R_1.program }
5. **R → PR R_1** { R.program:= R_1.program }
6. **R → λ** { R.program:= 0 }

7. **PP → program PPid ; D M2 Bloque ;** {
   if (Bloque.tipo = tipo_error) then Error (error_bloque)
   if (Bloque.tipoRet ≠ tipo_ok AND Bloque.tipoRet ≠ vacío)
   then Error (program_con_retorno)
   if (Bloque.exit > 0) then Error (exit_fuera_bucle)
   DestruirTS (TSL)
   TSActual:= TSG
   zona_decl:= true
   }

8. **PPid → Pid** {
```
    InsetarTipoTS (Pid.pos, vacío→vacío)
    InsertarEtiqTS (Pid.pos, "main")
}
```
9. **Pid → id** {
```
    TSL:= CrearTS ()
    TSActual:= TSL
    despl_local:= 0
    Pid.pos:= id.pos
}
```
10. **M2 → λ** { zona_decl:= false }
11. **PR → procedure PRidA ; D M2 Bloque ;** {
```
    if (Bloque.tipo = tipo_error) then Error (error_bloque)
    if (Bloque.tipoRet ≠ vacío AND Bloque.tipoRet ≠ tipo_ok)
    then Error (procedure_con_retorno)
    if (Bloque.exit > 0) then Error (exit_fuera_bucle)
    DestruirTS (TSL)
    TSActual:= TSG
    zona_decl:= true
}
```
12. **PRidA → Pid A** {
```
    InsertarTipoTS (Pid.pos, A.tipo→vacío);
    PRidA.et:= nueva_et ()
    InsertarEtiqTS (Pid.pos, PRidA.et);
}
```
13. **PF → function PFidAT ; D M2 Bloque ;** {
```
    if (Bloque.tipo = tipo_error) then Error (error_bloque)
    if (Bloque.tipoRet ≠ PFidAT.tipo AND Bloque.tipoRet ≠ tipo_ok)
    then Error (function_con_retorno_incorrecto)
    if (Bloque.exit > 0) then Error (exit_fuera_bucle)
    DestruirTS (TSL)
    TSActual:= TSG
    zona_decl:= true
}
```
14. **PFidAT → Pid A : T** {
```
    PFidAT.tipo:= T.tipo
    InsertarTipoTS (Pid.pos, A.tipo→T.tipo);
    PFidAT.et:= nueva_et()
    InsertarEtiqTS (Pid.pos, PFidAT.et);
}
```
15. **D → var id : T ; DD** {
```
    InsertarTipoTS (id.pos,T.tipo)
    if (TSActual = TSG)
    then{
            InsertarDespTS (id.pos, despl_global)
            despl_global:= despl_global + T.ancho
        }
    else{
            InsertarDespTS (id.pos, despl_local)
            despl_local:= despl_local + T.ancho
        }
}
```
16. **D → λ** { }
17. **DD → id : T ; DD** {
```
    InsertarTipoTS (id.pos,T.tipo)
    if (TSActual = TSG)
    then{
            InsertarDespTS (id.pos, despl_global)
            despl_global:= despl_global + T.ancho
        }
    else{
            InsertarDespTS (id.pos, despl_local)
            despl_local:= despl_local + T.ancho
        }
}
```
18. **DD → λ** { }

19. **T → boolean** { T.tipo:= lógico; T.ancho:= 1 }
20. **T → integer** { T.tipo:= entero; T.ancho:= 1 }
21. **T → string** { T.tipo:= cadena; T.ancho:= 64 }

22. **A → (X id : T AA)**
```
{   InsertarTipoTS (id.pos, T.tipo)
    InsertarDespTS (id.pos,despl_local)
    if (X.referencia)
    then{
            InsertaAtributoPasoParametrosTS (id.pos, "referencia")
            despl_local:= despl_local + 1
        }
    else{
            InsertaAtributoPasoParametrosTS (id.pos, "valor")
            despl_local:= despl_local + T.ancho
        }
    if (AA.tipo ≠ vacío)
    then A.tipo:= T.tipo x AA.tipo
    else A.tipo:= T.tipo
    A.long:= 1 + AA.long
}
```

23. **A → λ**
```
{   A.tipo:= vacío
    A.long:= 0
}
```

24. **AA → ; X id : T AA_1**
```
{   InsertarTipoTS (id.pos, T.tipo)
    InsertarDespTS (id.pos,despl_local)
    if (X.referencia)
    then{
            InsertaAtributoPasoParametrosTS (id.pos, "referencia")
            despl_local:= despl_local + 1
        }
    else{
            InsertaAtributoPasoParametrosTS (id.pos, "valor")
            despl_local:= despl_local + T.ancho
        }
    if (AA1.tipo ≠ vacío)
    then AA.tipo:= T.tipo x AA1.tipo
    else AA.tipo:= T.tipo
    AA.long:= 1 + AA1.long}
```

25. **AA → λ**
```
{   AA.tipo:= vacío
    AA.long:= 0
}
```

26. **Bloque → begin C end**
```
{   Bloque.tipo:= C.tipo
    Bloque.exit:= C.exit
    Bloque.tipoRet:= C.tipoRet
}
```

27. **C → B C_1**
```
{   C.tipo:= if (B.tipo = tipo_ok)
             then C1.tipo
             else tipo_error
    C.exit:= B.exit + C1.exit
    C.tipoRet:=  if (B.tipoRet = C1.tipoRet)
                 then B.tipoRet
                 else if (B.tipoRet = tipo_ok)
                 then C1.tipoRet
                 else if (C1.tipoRet = tipo_ok)
                 then B.tipoRet
                 else tipo_error
}
```

28. **C → λ**
```
{   C.tipo:= tipo_ok
    C.exit:= 0
    C.tipoRet:= tipo_ok
}
```

29. **B → if EE then S**{
```
B.tipo:= if (EE.tipo = lógico)
         then S.tipo
         else tipo_error
B.exit:= S.exit
B.tipoRet:= S.tipoRet
}
```

30. **EE → E** { EE.tipo:= E.tipo }

31. **B → S** {
```
B.tipo:= S.tipo
B.exit:= S.exit
B.tipoRet:= S.tipoRet
}
```

32. **B → if EE then Bloque ;** {
```
B.tipo:= if (EE.tipo = lógico)
         then Bloque.tipo
         else tipo_error
B.exit:= Bloque.exit
B.tipoRet:= Bloque.tipoRet
}
```

33. **B → if THEN else Bloque ;** {
```
B.tipo:= if (THEN.tipo = tipo_ok)
         then Bloque.tipo
         else tipo_error
B.exit:= THEN.exit + Bloque.exit
B.tipoRet:=  if (THEN.tipoRet = Bloque.tipoRet)
             then Bloque.tipoRet
             else if (THEN.tipoRet = tipo_ok)
             then Bloque.tipoRet
             else if (Bloque.tipoRet = tipo_ok)
             then THEN.tipoRet
             else tipo_error
}
```

34. **THEN → EE then Bloque ;**{
```
THEN.tipo:=  if (EE.tipo = lógico)
             then Bloque.tipo
             else tipo_error
THEN.exit:= Bloque.exit
THEN.tipoRet:= Bloque.tipoRet
}
```

35. **B → while M3 EE do Bloque ;** {
```
B.tipo:= if (EE.tipo = lógico)
         then Bloque.tipo
         else tipo_error
B.exit:= Bloque.exit
B.tipoRet:= Bloque.tipoRet
}
```

36. **M3 → λ** { }

37. **B → repeat M3 C until E ;** {
```
B.tipo:= if (E.tipo = lógico)
         then C.tipo
         else tipo_error
B.exit:= C.exit
B.tipoRet:= C.tipoRet
}
```

38. **B → loop M3 C end ;** {
```
B.tipo:= C.tipo
B.exit:= 0
if (C.exit ≠ 1) then Error (“dentro de loop debe haber 1 y solo 1 exit”)
B.tipoRet:= C.tipoRet
}
```

39. **B → FOR do Bloque ;**
```
{   B.tipo:= if (FOR.tipo = tipo_ok)
                then Bloque.tipo
                else tipo_error
    B.exit:= Bloque.exit
    B.tipoRet:= Bloque.tipoRet
}
```

40. **FOR → for id := E_1 to E_2**
```
{   FOR.tipo:=  if (E1.tipo = E2.tipo = BuscaTipoTS (id.pos) = entero)
                then tipo_ok
                else tipo_error
}
```

41. **B → case EXP of N O end;**
```
{   B.tipo:= if (EXP.tipo = entero AND N.tipo = O.tipo = tipo_ok)
                then tipo_ok
                else tipo_error
    B.exit:= N.exit + O.exit
    B.tipoRet:=  if (N.tipoRet = O.tipoRet)
                 then N.tipoRet
                 else if (N.tipoRet = tipo_ok)
                 then O.tipoRet
                 else if (O.tipoRet = tipo_ok)
                 then N.tipoRet
                 else tipo_error
}
```

42. **EXP → E** `{ EXP.tipo:= E.tipo }`

43. **N → N_1 VALOR : Bloque ;**
```
{   N.tipo:= if (N1.tipo = tipo_ok)
                then Bloque.tipo
                else tipo_error
    N.exit:= N1.exit + Bloque.exit
    N.tipoRet:=  if (Bloque.tipoRet = N1.tipoRet)
                 then Bloque.tipoRet
                 else if (N1.tipoRet = tipo_ok)
                 then Bloque.tipoRet
                 else if (Bloque.tipoRet = tipo_ok)
                 then N1.tipoRet
                 else tipo_error
}
```

44. **VALOR → entero** `{ }`

45. **N → λ**
```
{   N.tipo:= tipo_ok
    N.exit:= 0
    N.tipoRet:= tipo_ok
}
```

46. **O → otherwise : M3 Bloque ;**
```
{   O.tipo:= Bloque.tipo
    O.exit:= Bloque.exit
    O.tipoRet:= Bloque.tipoRet
}
```

47. **O → λ**
```
{   O.tipo:= tipo_ok
    O.exit:= 0
    O.tipoRet:= tipo_ok
}
```

48. **S → write LL ;**
```
{   S.tipo:= tipo_ok
    if (LL.tipo ≠ vacío) {
        for i:= 1 to LL.long
            if (LL.tipo[i] ≠ entero AND LL.tipo[i] ≠ cadena)
            then S.tipo:= tipo_error
        }
    S.exit:= 0
    S.tipoRet:= tipo_ok
}
```

49. **S → writeln LL ;** {
```
    S.tipo:= tipo_ok
    if (LL.tipo ≠ vacío) {
        for i:= 1 to LL.long
            if (LL.tipo[i] ≠ entero AND LL.tipo[i] ≠ cadena)
            then S.tipo:= tipo_error
        }
    S.exit:= 0
    S.tipoRet:= tipo_ok
}
```
50. **S → read (V) ;** {
```
    S.tipo:= tipo_ok
    for i:= 1 to V.long {
        if (V.tipo[i] ≠ entero AND V.tipo[i] ≠ cadena)
        then S.tipo:= tipo_error
    }
    S.exit:= 0
    S.tipoRet:= tipo_ok
}
```
51. **S → id := E ;** {
```
    S.tipo:= if (BuscaTipo (id.pos) = E.tipo ≠ tipo_error)
             then tipo_ok
             else tipo_error
    S.exit:= 0
    S.tipoRet:= tipo_ok
}
```
52. **S → id LL ;** {
```
    S.tipo:= if (buscaTipoTS (id.pos) = LL.tipo→vacío )
             then tipo_ok
             else tipo_error
    S.exit:= 0
    S.tipoRet:= tipo_ok
}
```
53. **S → return Y ;** {
```
    S.tipo:= if (Y.tipo ≠ tipo _error)
             then tipo_ok
             else tipo_error
    S.exit:= 0
    S.tipoRet:= Y.tipo
// tipoRet puede ser vacío (para procedimientos), un tipo (para funciones) o tipo_ok (cuando no hay return)
}
```
54. **S → exit when E ;** {
```
    S.tipo:= if (E.tipo = lógico)
             then tipo_ok
             else tipo_error
    S.exit:= 1
    S.tipoRet:= tipo_ok
}
```
55. **LL → (L)** {
```
    LL.tipo:= L.tipo
    LL.long:= L.long
}
```
56. **LL → λ** {
```
    LL.tipo:= vacío
    LL.long:= 0
}
```
57. **L → E Q** {
```
    L.tipo:= if (Q.tipo = vacío)
             then E.tipo
             else E.tipo x Q.tipo
    L.long:= 1 + Q.long
}
```
58. **Q → , E Q_1** {
```
    Q.tipo:= if (Q1.tipo = vacío)
             then E.tipo
             else E.tipo x Q1.tipo
    Q.long:= 1 + Q1.long
}
```
59. **Q → λ** {
```
    Q.tipo:= vacío
    Q.long:= 0
}
```

60. **V → id W** { V.tipo:= if (W.tipo = vacío)
then buscaTipoTS (id.pos)
else buscaTipoTS (id.pos) x W.tipo
V.long:= 1 + W.long
}

61. **W → , id W_1** { W.tipo:= if (W_1.tipo = vacío)
then buscaTipoTS (id.pos)
else buscaTipoTS (id.pos) x W_1.tipo
W.long:= 1 + W_1.long
}

62. **W → λ** { W.tipo:= vacío
W.long:= 0
}

63. **Y → E** { Y.tipo:= E.tipo }

64. **Y → λ** { Y.tipo:= vacío }

65. **E → E_1 or F** { E.tipo:= if (E_1.tipo = F.tipo = lógico)
then lógico
else tipo_error
}

66. **E → E_1 xor F** { E.tipo:= if (E_1.tipo = F.tipo = lógico)
then lógico
else tipo_error
}

67. **E → F** { E.tipo:= F.tipo }

68. **F → F_1 and G** { F.tipo:= if (F_1.tipo = G.tipo = lógico)
then lógico
else tipo_error
}

69. **F → G** { F.tipo:= G.tipo }

70. **G → G_1 = H** { G.tipo:= if (G_1.tipo = H.tipo = entero)
then lógico
else tipo_error }

71. **G → G_1 <> H** { G.tipo:= if (G_1.tipo = H.tipo = entero)
then lógico
else tipo_error
}

72. **G → G_1 > H** { G.tipo:= if (G_1.tipo = H.tipo = entero)
then lógico
else tipo_error
}

73. **G → G_1 >= H** { G.tipo:= if (G_1.tipo = H.tipo = entero)
then lógico
else tipo_error
}

74. **G → G_1 < H** { G.tipo:= if (G_1.tipo = H.tipo = entero)
then lógico
else tipo_error }

75. **G → G_1 <= H** { G.tipo:= if (G_1.tipo = H.tipo = entero)
then lógico
else tipo_error
}

76. **G → H** { G.tipo:= H.tipo }

77. **H → H_1 + I** { H.tipo:= if (H_1.tipo = I.tipo ∈ {entero, cadena})
then H_1.tipo
else tipo_error
}

78. **H → H_1 - I** { H.tipo:= if (H_1.tipo = I.tipo = entero)
then entero
else tipo_error
}
79. **H → I** { H.tipo:= I.tipo }

80. **I → I_1 * J** { I.tipo:= if (I_1.tipo = J.tipo = entero)
then entero
else tipo_error
}
81. **I → I_1 / J** { I.tipo:= if (I_1.tipo = J.tipo = entero)
then entero
else tipo_error
}
82. **I → I_1 mod J** { I.tipo:= if (I_1.tipo = J.tipo = entero)
then entero
else tipo_error }
83. **I → J** { I.tipo:= J.tipo }

84. **J → J_1 ** K** { J.tipo:= if (J_1.tipo = K.tipo = entero)
then entero
else tipo_error }
85. **J → K** { J.tipo:= K.tipo }

86. **K → not K1**{ K.tipo:= if (K1.tipo = lógico)
then lógico
else tipo_error
}
87. **K → + K1** { K.tipo:= if (K1.tipo = entero)
then entero
else tipo_error
}
88. **K → - K1** { K.tipo:= if (K1.tipo = entero)
then entero
else tipo_error
}
89. **K → Z** { K.tipo:= Z.tipo }

90. **Z → entero** { Z.tipo:= entero }
91. **Z → cadena** { Z.tipo:= cadena }
92. **Z → true** { Z.tipo:= lógico }
93. **Z → false** { Z.tipo:= lógico }
94. **Z → id LL** { id.tipo:= BuscaTipoTS (id.pos)
Z.tipo:= if (id.tipo = LL.tipo→t AND t ≠ vacío)
then t
else if (id.tipo ∈ {entero, lógico, cadena} AND LL.tipo = vacío)
then id.tipo
else tipo_error
}
95. **Z → (E)** { Z.tipo:= E.tipo }
96. **Z → Z_1 in (L)** { Z.tipo:= if (Z_1.tipo = entero)
then lógico
else tipo_error
for i = 1 to L.long
if (L.tipo[i] ≠ entero)
then Z.tipo:= tipo_error
}
97. **Z → max (L)** { Z.tipo:= entero
for i = 1 to L.long
if (L.tipo[i] ≠ entero)
then Z.tipo:= tipo_error
}

98. **Z → min (L)** { Z.tipo:= entero
for i = 1 to L.long
if (L.tipo[i] ≠ entero)
then Z.tipo:= tipo_error
}

99. **X → var** { X.referencia:= true }

100. **X → λ** { X.referencia:= false }