PRÁCTICA DE PROCESADORES DE LENGUAJES

Curso 2013 – 2014

Entrega de Junio

APELLIDOS Y NOMBRE: **Remirez Ruiz, Paula**

IDENTIFICADOR: **premirez1**

DNI: **72680972Y**

CENTRO ASOCIADO MATRICULADO: **Pamplona - NAVARRA**

CENTRO ASOCIADO DE LA SESIÓN DE CONTROL: **Pamplona**

MAIL DE CONTACTO: **pauremirez@gmail.com**

TELÉFONO DE CONTACTO: **657603358**

GRUPO (A ó B): **A**

INDICE

1 Cambios realizados en el analizador léxico y sintáctico

1.1 El analizador léxico

No se han realizado cambios en el analizador léxico en esta segunda parte. El analizador sigue quedando de la siguiente forma:

[…]

```
//Macros necesarias (Directivas)

LETRA = [A-Za-z]
DIGITO = [0-9]
ESPACIO = [\ \t\r\n]+
CARACTERESCADENA = [^\"]*
ENTERO = {DIGITO}+
ID = {LETRA}({LETRA}|{DIGITO})*
COMENTARIOLINEA=--.*\r\n

%%

<YYINITIAL>
{
"array"{Token token = new Token (sym.ARRAY);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
"begin"{Token token = new Token (sym.BEGIN);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
"Boolean"{Token token = new Token (sym.BOOLEAN);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
"constant"{Token token = new Token (sym.CONSTANT);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
"else"{Token token = new Token (sym.ELSE);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
"end"{Token token = new Token (sym.END);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
"false"{Token token = new Token (sym.FALSE);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
"function"{Token token = new Token (sym.FUNCTION);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
"if"{Token token = new Token (sym.IF);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
"Integer"{Token token = new Token (sym.INTEGER);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“is"{Token token = new Token (sym.IS);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“loop"{Token token = new Token (sym.LOOP);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“of"{Token token = new Token (sym.OF);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“or"{Token token = new Token (sym.OR);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“out"{Token token = new Token (sym.OUT);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“procedure"{Token token = new Token (sym.PROCEDURE);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“Put_line"{Token token = new Token (sym.PUTLINE);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“return"{Token token = new Token (sym.RETURN);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“then"{Token token = new Token (sym.THEN);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“True"{Token token = new Token (sym.TRUE);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“type"{Token token = new Token (sym.TYPE);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“while"{Token token = new Token (sym.WHILE);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}

“\""{Token token = new Token (sym.COMILLASDOBLES);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“("{Token token = new Token (sym.PARENTESISIZQ);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“)"{Token token = new Token (sym.PARENTESISDER);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“.."{Token token = new Token (sym.PUNTOPUNTO);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“--"{Token token = new Token (sym.INICIOCOMENTARIO);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
```

```
“\n"{Token token = new Token (sym.SALTOLINEA);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“,"{Token token = new Token (sym.COMA);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“;"{Token token = new Token (sym.PUNTOYCOMA);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
“:"{Token token = new Token (sym.DOSPUNTOS);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}

"-"{Token token = new Token (sym.MENOS);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
"+"{Token token = new Token (sym.MAS);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
"<"{Token token = new Token (sym.MENORQUE);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
">"{Token token = new Token (sym.MAYORQUE);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
"/="{Token token = new Token (sym.DISTINTOQUE);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
"="{Token token = new Token (sym.IGUALQUE);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}

"and"{Token token = new Token (sym.AND);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
":="{Token token = new Token (sym.ASIGNACION);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
"."{Token token = new Token (sym.PUNTO);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}

\"{CARACTERESCADENA}\"
{Token token = new Token (sym.CARACTERESCADENA);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ()); return token;}
{ID}
{Token token = new Token (sym.ID);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
{ENTERO}
{Token token = new Token (sym.ENTERO);token.setLine (yyline + 1);token.setColumn (yycolumn + 1);token.setLexema (yytext ());return token;}
{ESPACIO}
{ } //ESPACIOS EN BLANCO (NO HACEMOS NADA) //
{COMENTARIOLINEA}
{ } //NO HACEMOS NADA
[^]
{LexicalError error = new LexicalError ();error.setLine (yyline + 1);error.setColumn (yycolumn + 1);error.setLexema (yytext
());lexicalErrorManager.lexicalError (error);} }//Fin YYNITIAL
```

1.2 El analizador sintáctico

No se han realizado cambios en el analizador sintáctico en cuanto a la precedencia de operadores, ni en la declaración de no terminales en esta segunda parte.

2. El analizador semántico y la comprobación de tipos

En el paquete *compiler.syntax.nonTerminal* he creado varias clases de apoyo que dan soporte al analizador semático que heredan de la clase nonTerminal:

BloqueSentencias	Clase para agrupar código intermedio para un grupo de sentencias
Expresion	Clase para código intermedio de expresiones simples
ListaIdentificadores	Clase de apoyo para contener una lista de identificadores
ListaObjetos	Clase de apoyo para contener una lista de variables
Parámetro	Clase de apoyo para los parámetros de funciones y procedimientos
ParametrosFormales	Clase de apoyo para los parámetros formales
Var	Clase de apoyo para la declaración de variables

La comprobación de tipos se realiza para todas las expresiones de la gramática y estructuras de la gramática.

2.1 Descripción de la tabla de símbolos

Cuando se crea un ámbito se asocia ese ámbito a una tabla de símbolos, proporcionada por la clase SymbolTable y una tabla de tipo, proporcionada por la clase TableType.

En la práctica se han utilizado las siguientes clases para el apoyo de la declaración de los símbolos utilizados en el código fuente (constantes, variables….):

Tabla de símbolos:

SymbolConstant	Clase de apoyo para la declaración de constantes
SymbolFunction	Clase de apoyo para la declaración de funciones
SymbolParameter	Clase de apoyo para la declaración de parametros
SymbolProcedure	Clase de apoyo para la declaración de procedimientos
SymbolTable	Clase de apoyo para la declaración de los diferentes símbolos
SymbolVariable	Clase de apoyo para la declaración de variables

En la práctica se incluyen los tipos primitivos que se insertan a abrir el scope del programa principal y los compuestos que se van insertando en el ámbito en el que aparecen en el código fuente.

Tabla de tipos:

TypeArray	Define el tipos simple ARRAY, para representar arrays
TypeBoolean	Define el tipos simple BOOLEAN, para representar booleanos
TypeFunction	Clase para la definición de funciones. Incluye todos los atributos necesarios para la declaración de las funciones
TypeInteger	Define el tipos simple INTEGER, para representar enteros
TypeProcedure	Clase para la definición de procedimientos. Incluye todos los atributos necesarios para la declaración de los procedimientos
TypeSimple	Crea la clase de la que heredan los tipos primitivos del programa
TypeTable	Clase de apoyo a la declaración de tipos

3. Generación de código intermedio

Para dar soporte a la generación de código intermedio se han utilizado las siguientes clases, alojadas en el paquete *compiler.intermediate*:

Label	Da soporte a la generación de etiquetas
Temporal	Gestión de los temporales
Value	Gestión de los valores
Variable	Gestión de las variables para pasarlas a ensamblador

A continuación se detalla la estructura utilizada, y para poder clarificar su utilización, aunque resulta un poco extenso, se detalla la generación de un programa real en HAda **\test\testA.ha** con sus correspondientes cuádruplas.

3.1 Descripción de la estructura utilizada

Cuadrupla	Resultado	Operador1	Operador2	Descripción
INICIO_PROGRAMA	Variable	Valor		Inicio del programa principal. La variable contiene la tabla de desplazamiento por scope del programa y el valor de desplazamiento del programa principal.
FIN_PROGRAMA				Fin del programa principal.
CADENA	cadena	contador		Cadena de texto del programa. Se añaden al final con la instrucción DATA. El contador indica una numeración consecutiva.
SUB	Temporal	Op1	Op2	Resta el Op1 menos el Op2 y deja el resultado en el temporal
MUL	Temporal	Op1	Op2	Multiplica el Op1 y el Op2 y deja el resultado en el temporal
ADD	Temporal	Op1	Op2	Suma el Op1 y el Op2 y deja el resultado en el temporal
CMP	Op1	Op2		Compara el Op1 y el Op2. Los operadores pueden ser Variables, Temporales o Valores.
MV	Variable/Temporal	Op1		Mueve el Op1 a la Variable/Temporal
INL	Label			Crea la etiqueta que indica Label
BZ	Label			Salto si Label = 0
BNZ	Label			Salto si Label <>0
BN	Label			Salto si Label es negativo
BR	Label			Salto incondicional a la etiqueta que indica Label
BP	Label			Salto si Label es positivo
CALL	Variable	Nivel	Temporal/Value	Llamada a programa especificado en variable con el nivel de anidamiento especifico de cada nivel y un temporal para el retorno. Si no hay retorno (procedimiento), se pasa el Valor 0.
ARGUMENTO	Resultado	Valor		Argumento por valor de llamadas en funciones/procedimientos. El resultado contiene también el desplazamiento del parámetro.
ARGUMENTO_REF	Resultado	Valor		Argumento por referencia de llamadas en funciones/procedimientos. El resultado contiene también el desplazamiento del parámetro.
INICIO_ARGUMENTOS				Inicio de los argumentos de los procedimientos/funciones. Sirve para llevar el contador en programas.
RET	Variable	Op1		Valor de retorno de una función.
INICIO_SUBPROG	Label			Determina el inicio de un subprograma.

FIN_SUBPROG	Label	Label	Nivel	Etiqueta de fin de definición de subprograma y la etiqueta para retorno del subprograma. El nivel indica el anidamiento.
WRSTR	Cadena			Escritura de una cadena
WRINT_ARRAY	Valor			Escritura de un valor procedente de un array
WRINT	Valor			Escritura de un valor
NOP				No operación.
ASIGN_VECTOR	Variable	Valor	Valor	Asigna en la variable una expresión a la posición correcta del array. Contiene el desplazamiento correspondiente.
VALOR_VECTOR	Variable	Valor		Obtiene el valor de una posición concrete del array.

3.2 Ejemplo completo de código fuente a código intermedio

CODIGO FUENTE	**CUÁDRUPLA ASOCIADA**
`procedure testA() is`	`[INICIO_PROGRAMA testA, 9, null]`
`-- constantes y variables globales` `A: constant := 1;`	

`d: Integer;`
`e: Integer;`
`f: Integer;`	

`function resta (a : out Integer; b: out Integer) return Integer is` `begin`	`[INICIO_SUBPROG L_0, null, null]`
`  return a-b;`	`[SUB T_0, a, b]` `[RET T_0, null, null]`
`end resta;`	`[FIN_SUBPROG L_0, L_1, 0]`
`-- procedimiento principal`	
`begin`	
`  e := A;` `  f := 4;`	`[MV e, 1, null]` `[MV f, 4, null]`

`  d := resta (f,e);`	`[INICIO_ARGUMENTOS null, null, null]`
	`[ARGUMENTO f, 5, null]`
	`[ARGUMENTO e, 6, null]`
	`[CALL resta, 0, T_0]`
	`[MV d, T_0, null]`

`  Put_line("resultado:");`	`[WRSTR cadena1, null, null]`
`  Put_line(d);   -- Debe mostar un 3`	`[WRINT d, null, null]`

`end testA;`	`[FIN_PROGRAMA null, null, null]`
	`[CADENA cadena0, "\n", ]`
	`[CADENA cadena1, "resultado:", ]`

4. Generación de código final

Partiendo del código intermedio generado y con la ayuda de la clase ENS2001Environment.java del paquete *compiler.code*, traducimos las cuádruplas a código ensamblador.

Se utiliza para este apartado el mismo programa utilizado para generar las cuádruplas del código intermedio (**test\testA.ha**) – ver en página siguiente -

4.1 Ejemplo de código completo de código intermedio a ensamblador

CUÁDRUPLA ASOCIADA	CODIGO ENSAMBLADOR
[INICIO_PROGRAMA testA, 9, null]	; INICIO PROGRAMA PRINCIPAL HAda RES 21 MOVE #65535 , .SP MOVE .SP , .IY SUB .SP , #9 MOVE .A , .SP MOVE .IY , /0
[INICIO_SUBPROG L_0, null, null]	;-- Definicion FUNCION/PROCEDIMIENTO BR /FIN_L_0 L_0 :

[SUB T_0, a, b]	SUB #-5[.IY], #-6[.IY]
	MOVE .A, #-7[.IY]
	MOVE .A, .R9

[RET T_0, null, null]	; Traducir Quadruple - [RET T_0, null, null] MOVE #-7[.IY] , #-3[.IX] MOVE #-1[.IY] , .PC L_1 :
[FIN_SUBPROG L_0, L_1, 0]	; Retorno Argumentos REFERENCIA MOVE #-4[.IY] , .R5 MOVE .R5 , .PC REF_L_0: ; Retorno Subprograma MOVE #-1[.IY] , .R7 MOVE #-2[.IY] , /0 MOVE .IY , .SP MOVE .IX , .IY MOVE #-3[.IY] , .IX MOVE .R7 , .PC MOVE .A, #-5[.IY] FIN_L_0 :

[MV e, 1, null]
[MV f, 4, null]
	; Traducir Quadruple - [MV e, 1, null]
	MOVE #1, #-6[.IY]
	; Traducir Quadruple - [MV f, 4, null]
	MOVE #4, #-7[.IY]

[INICIO_ARGUMENTOS null, null, null] [ARGUMENTO f, 5, null] [ARGUMENTO e, 6, null] [CALL resta, 0, T_0]	; INICIO ARGUMENTOS FIN ; Cargado argumento Quadruple - [ARGUMENTO f, 5, null] ; Cargado argumento Quadruple - [ARGUMENTO e, 6, null] ; Llamada Funcion Quadruple - [CALL resta, 0, T_0] MOVE .SP , .IX MOVE #-7[.IY] , #-5[.IX] MOVE #-6[.IY] , #-6[.IX] MOVE /0 , #-2[.IY] MOVE .IY , /0 MOVE .IY , .IX MOVE .SP , .IY SUB .SP , #8 MOVE .A , .SP MOVE #RET_L_2 , #-1[.IY] MOVE .IX , #-3[.IY] MOVE #REF_L_2 , #-4[.IY] BR /L_0 REF_L_2: BR /REF_L_0 RET_L_2: MOVE .R9 , #-8[.IY]
[MV d, T_0, null] [WRSTR cadena1, null, null]	; Traducir Quadruple - [MV d, T_0, null] MOVE #-8[.IY], #-5[.IY] WRSTR /cadena1 WRCHAR #10 WRCHAR #13
[WRINT d, null, null] [FIN_PROGRAMA null, null, null]	WRINT #-5[.IY] WRCHAR #10 WRCHAR #13 HALT
[CADENA cadena0, "\n",] [CADENA cadena1, "resultado:",]	; Inicio Cadenas de Texto cadena0: DATA "\n" cadena1: DATA "resultado:"

5. Conclusiones

La primera parte de la práctica me pareció más sencilla y a la vez más guiada que esta segunda parte. Para su realización existían un par de video-tutoriales y más explicación para poder realizarla.

Esta segunda parte es muchísimo más extensa y compleja. Me he visto en varias veces sobrepasada por la cantidad de trabajo que conlleva y por la desinformación que tenía.

Creo que he podido resolver muchos de los problemas por el foro de la asignatura (tanto por las respuestas proporcionadas por los compañeros como por el equipo docente), pero si que es cierto que no me parece el canal más rápido para solucionarles, ya que ha habido veces que he estado parada varios días sin poder resolver ciertas cosas porque no obtenía respuesta a mis preguntas.

A titulo personal creo que se podrían facilitar más partes de código ya implementadas. Esto reduciría la carga de trabajo y podría orientar un poco a futuros alumnos cómo afrontar la práctica.

6. Gramática

```
//--------------------------
// Declaracion de terminales
//--------------------------
terminal Token MAS;
terminal Token ARRAY;
terminal Token BEGIN;
terminal Token BOOLEAN;
terminal Token CONSTANT;
terminal Token ELSE;
terminal Token END;
terminal Token FALSE;
terminal Token FUNCTION;
terminal Token IF;
terminal Token INTEGER;
terminal Token IS;
terminal Token LOOP;
terminal Token OF;
terminal Token OR;
terminal Token OUT;
terminal Token PROCEDURE;
terminal Token PUTLINE;
terminal Token RETURN;
terminal Token THEN;
terminal Token TRUE;
terminal Token TYPE;
terminal Token WHILE;
terminal Token COMILLASDOBLES;
terminal Token PARENTESISIZQ;
terminal Token PARENTESISDER;
terminal Token INICIOCOMENTARIO;
terminal Token SALTOLINEA;
terminal Token COMA;
terminal Token PUNTOYCOMA;
terminal Token DOSPUNTOS;
terminal Token MENOS;
terminal Token MENORQUE;
terminal Token MAYORQUE;
terminal Token IGUALQUE;
terminal Token DISTINTOQUE;
terminal Token AND;
terminal Token ASIGNACION;
terminal Token PUNTO;
```

```
terminal Token ID; //Identificador
terminal Token ENTERO;
terminal Token PUNTOPUNTO;
terminal Token CARACTERESCADENA;

//----------------------------
// Declaracion de no terminales
//----------------------------
// no modificar los propuestos

non terminal Axiom                          axiom;
non terminal                                program;
non terminal                                seccionTipos;
non terminal                                seccionVariables;
non terminal BloqueSentencias               seccionSubProgramas;
non terminal                                seccionConstantesSimbolicas;
non terminal                                declaracionConstanteSimbolica;
non terminal                                declaracionTipo;
non terminal                                declaracionVariable;
non terminal BloqueSentencias               declaracionSubPrograma;
non terminal BloqueSentencias               funcion;
non terminal BloqueSentencias               procedimiento;
non terminal ListaObjetos                   parametros;
non terminal ListaObjetos                   lista_argumentos;
non terminal ListaObjetos                   lista_parametros;
non terminal ListaObjetos                   decVariable;
non terminal BloqueSentencias               listaSentencias;
non terminal BloqueSentencias               sentencia;
non terminal BloqueSentencias               sentenciaProcedimiento;
non terminal BloqueSentencias               sentenciaPutLine;
non terminal BloqueSentencias               sentenciaWhile;
non terminal BloqueSentencias               sentenciaIf;
non terminal BloqueSentencias               sentenciaAsignacion;
non terminal BloqueSentencias               idTipos;
non terminal Expresion                      expresion;
non terminal Token                          tipoBooleano;
non terminal                                vacio;
non terminal BloqueSentencias               cuerpoFuncion;
non terminal BloqueSentencias               cabecera;
non terminal BloqueSentencias               cuerpo;
non terminal Expresion                      sentenciaFuncion;
non terminal ListaObjetos                   lista_parametros_llamada;
non terminal                                tipos;
non terminal ListaObjetos                   parametro;
non terminal                                modo;
non terminal BloqueSentencias               procedure;
non terminal Expresion                      vector;


//----------------------------------------
// Declaracion de relaciones de precedencia
//----------------------------------------
precedence left MAS, MENOS;
precedence nonassoc ASIGNACION;
precedence left MENORQUE, MAYORQUE, DISTINTOQUE, IGUALQUE;
precedence left AND, OR;
precedence left PARENTESISIZQ, PARENTESISDER, PUNTO;
precedence right ELSE;


//-----------------------------------
// Axioma
//-----------------------------------
start with program;
program::=
  {:
        syntaxErrorManager.syntaxInfo ("Starting parsing...");
  :}
  axiom:ax
  {:
       syntaxErrorManager.syntaxInfo ("Parsing process ended.");
  :};
```

```
// ----------------------------------------
// Declaracion de las reglas de producción
// ----------------------------------------
axiom::= PROCEDURE ID PARENTESISIZQ PARENTESISDER IS cabecera cuerpo | PROCEDURE ID PARENTESISIZQ PARENTESISDER IS cuerpo;
cabecera::= seccionConstantesSimbolicas seccionTipos seccionVariables seccionSubProgramas |
       seccionConstantesSimbolicas seccionVariables seccionSubProgramas | seccionConstantesSimbolicas seccionTipos seccionSubProgramas |
       seccionConstantesSimbolicas seccionTipos seccionVariables | seccionConstantesSimbolicas seccionSubProgramas |
       seccionConstantesSimbolicas seccionVariables | seccionConstantesSimbolicas | seccionTipos seccionVariables seccionSubProgramas |
       seccionTipos seccionVariables | seccionTipos seccionSubProgramas | seccionTipos |
       seccionVariables seccionSubProgramas | seccionVariables | seccionSubProgramas;
cuerpo::= BEGIN listaSentencias END ID PUNTOYCOMA | BEGIN END ID PUNTOYCOMA;
seccionConstantesSimbolicas::= declaracionConstanteSimbolica | seccionConstantesSimbolicas declaracionConstanteSimbolica ;
seccionTipos::= declaracionTipo | seccionTipos declaracionTipo;
seccionVariables::= declaracionVariable | seccionVariables declaracionVariable;
seccionSubProgramas::= declaracionSubPrograma | seccionSubProgramas declaracionSubPrograma;
declaracionConstanteSimbolica::= ID DOSPUNTOS CONSTANT ASIGNACION tipoPrimitivo PUNTOYCOMA;
declaracionTipo::= TYPE ID IS ARRAY PARENTESISIZQ ENTERO PUNTOPUNTO ENTERO PARENTESISDER OF tipos PUNTOYCOMA;
declaracionVariable::= idParametros tipos PUNTOYCOMA;
declaracionSubPrograma::= procedimiento | funcion;
funcion::= FUNCTION ID PARENTESISIZQ parametros PARENTESISDER RETURN tipos IS cabecera cuerpoFuncion |
       FUNCTION ID PARENTESISIZQ PARENTESISDER RETURN tipos IS cabecera cuerpoFuncion |
       FUNCTION ID PARENTESISIZQ parametros PARENTESISDER RETURN tipos IS cuerpoFuncion |
       FUNCTION ID PARENTESISIZQ PARENTESISDER RETURN tipos IS cuerpoFuncion;
cuerpoFuncion::= BEGIN listaSentencias PUNTOYCOMA END ID PUNTOYCOMA | BEGIN PUNTOYCOMA END ID PUNTOYCOMA;
procedimiento::= PROCEDURE ID PARENTESISIZQ parametros PARENTESISDER IS cabecera cuerpo | PROCEDURE ID PARENTESISIZQ parametros PARENTESISDER IS cuerpo;
listaSentencias::= sentencia | listaSentencias sentencia ;
sentencia::=sentenciaAsignacion | sentenciaIf | sentenciaWhile | sentenciaPutLine | sentenciaProcedimiento | RETURN expresion;
sentenciaPutLine::= PUTLINE PARENTESISIZQ parametroPutLine PARENTESISDER PUNTOYCOMA;
parametroPutLine::= cadenaCaracteres | expresion;
cadenaCaracteres::= CARACTERESCADENA;
sentenciaWhile::= WHILE expresion LOOP listaSentencias END LOOP PUNTOYCOMA;
sentenciaIf::= IF expresion THEN listaSentencias END IF PUNTOYCOMA | IF expresion THEN listaSentencias ELSE listaSentencias END IF PUNTOYCOMA;
sentenciaAsignacion::= idTipos ASIGNACION expresion PUNTOYCOMA;
idTipos::= ID | ID PUNTO ID | ID PUNTO ID PUNTO ID | vector;
vector::= ID PARENTESISIZQ ENTERO PARENTESISDER;
sentenciaFuncion ::= ID PARENTESISIZQ lista_parametros_llamada PARENTESISDER PUNTOYCOMA;
lista_parametros_llamada::= expresion | expresion COMA lista_parametros_llamada;
sentenciaProcedimiento::= idTipos ASIGNACION ID PARENTESISIZQ lista_parametros_llamada PARENTESISDER PUNTOYCOMA |
                          ID PARENTESISIZQ lista_parametros_llamada PARENTESISDER PUNTOYCOMA;
expresion::= expresion MENOS expresion | expresion MAS expresion | expresion OR expresion | expresion AND expresion | expresion MAYORQUE expresion |
             expresion MENORQUE expresion | expresion IGUALQUE expresion | expresion DISTINTOQUE expresion | vector |
             PARENTESISIZQ expresion PARENTESISDER | ID | ENTERO | ID PUNTO ID | ID PUNTO ID PUNTO ID | sentenciaFuncion | tipoBooleano;
tipoPrimitivo::= ENTERO | tipoBooleano;
tipoBooleano::= TRUE | FALSE;
vacio::=;
tipos::= INTEGER | BOOLEAN | ID;
idParametros::= ID DOSPUNTOS | ID COMA idParametros;
parametros::= parametro | parametro PUNTOYCOMA parametros;
parametro::= idParametros modo tipos;
modo::= OUT | vacio;
```