Sara Grimaldos Rodríguez

UO251782@uniovi.es

Reglas Lex

```
Identificador = [a-zA-Zñá-úÁ-Ú][a-zA-Zñá-úÁ-Ú0-9]*
ConstanteEntera = [0-9]*
Real = [.][0-9]+|[0-9]+[.][0-9]*
ConstanteReal = {Real}|{Real}E-[0-9]+|[0-9]+e[+][0-9]+|[0-9]+e-[0-9]+
CodigoASCII = [']\\[0-9]*[']
```

Gramática

```
// * Declaraciones Yacc
%token CTE_ENTERA
%token DISTINTO
%token MAIN
%token IF
%token CTE_CARACTER
%token ID
%token MENOR_IGUAL
%token ELSE
%token WRITE
%token VOID
%token MAYOR_IGUAL
%token IGUAL_IGUAL
%token CTE_REAL
%token STRUCT
%token WHILE
%token FUNC
%token FLOAT32
%token INT
%token RETURN
%token CHAR
%token AND
%token READ
%token VAR
%token OR
%token MAS_MAS
%token MENOS_MENOS
%token MAS_IGUAL
%token MENOS_IGUAL
%token MUL_IGUAL
%token DIV_IGUAL

%right '='
%left AND OR
%left '>' MAYOR_IGUAL '<' MENOR_IGUAL DISTINTO IGUAL_IGUAL
%left '+' '-'
%left '*' '/' '%'
%right '!'
%right MENOSUNARIO
%nonassoc '[' ']'
%left '.'
%nonassoc '(' ')'
```

```
%%
// * Gramática y acciones Yacc

programa: lista_definiciones FUNC MAIN '(' ')' '{' cuerpo '}'
         ;

lista_definiciones: /* vacio */
                   | lista_definiciones definición
                   ;

definicion: definicion_funcion
            | definicion_variable
            ;

// * Expresiones

lista_expresiones: /* vacio */
                   | lista_expresionesP
                   ;

lista_expresionesP: expresion
                  | lista_expresionesP ',' expresion
                  ;

expresion: CTE_ENTERA
         | CTE_REAL
         | CTE_CARACTER
         | ID
         | '(' expresion ')'
         | expresion '.' ID
         | expresion '[' expresion ']'
         | '-' expresion %prec MENOSUNARIO
         | '!' expresion
         | expresion '*' expresion
         | expresion '/' expresion
         | expresion '%' expresion
         | expresion '+' expresion
         | expresion '-' expresion
         | expresion '>' expresion
         | expresion MAYOR_IGUAL expresion
         | expresion '<' expresion
         | expresion MENOR_IGUAL expresion
         | expresion DISTINTO expresion
         | expresion IGUAL_IGUAL expresion
         | expresion AND expresion
         | expresion OR expresion
         | tipo '(' expresion ')'
         | ID '(' lista_expresiones ')'
         ;


// * Sentencias

lista_sentencias: /* vacio */
                   | lista_sentencias sentencia
                   ;
```

```
sentencia: sentencia_if
             | sentencia_while
             | sentencia_write
             | sentencia_read
             | sentencia_asignacion
             | sentencia_return
             | sentencia_invocacion
             | sentencia_modificarValor
             | sentencia_modificarValorConcreto
             ;


sentencia_asignacion: expresion '=' expresion ';'
                    ;

sentencia_if: IF expresion '{' lista_sentencias '}'
            | IF expresion '{' lista_sentencias '}' ELSE '{' lista_sentencias '}'
            ;


sentencia_while: WHILE expresion '{' lista_sentencias '}'
               ;


sentencia_write: WRITE '(' lista_expresiones ')' ';'
               ;

sentencia_read: READ '(' lista_expresiones ')' ';'
              ;

sentencia_return: RETURN expresion ';'
               ;

sentencia_invocacion: ID '(' lista_expresiones ')' ';'
                    ;

sentencia_modificarValor: expresion MAS_MAS  ';'
                        | expresion MENOS_MENOS   ';'
                        ;

sentencia_modificarValorConcreto: expresion MAS_IGUAL expresion  ';'
                                | expresion MENOS_IGUAL   expresion ';'
                                | expresion MUL_IGUAL expresion ';'
                                | expresion DIV_IGUAL expresion ';'
                                ;


// * Definicion variable

lista_variables: /* vacio */
               |lista_variables definicion_variable
               ;

definicion_variable: VAR identificadores tipo ';'
```

```
                  ;

identificadores: ID
               | identificadores ',' ID
               ;

campo: identificadores tipo ';'
      ;

lista_campos: campo
            ;


tipo: INT
      | FLOAT32
      | CHAR
      | '[' CTE_ENTERA ']' tipo
      | STRUCT '{' lista_campos '}'
      ;


// * Definicion funcion

definicion_funcion: FUNC ID '(' lista_parametros ')' retorno '{' cuerpo '}'
                  ;

cuerpo: lista_variables lista_sentencias
      ;

tipoSimple: INT
          | FLOAT32
          | CHAR
          ;


retorno: /* vacio */
         | tipoSimple
         ;

lista_parametros: /* vacio */
                | lista_parametrosP
                ;

lista_parametrosP: ID tipoSimple
                 | lista_parametrosP ',' ID tipoSimple
                 ;
```

Plantillas de generación de código

```
EJECUTAR [[Programa: Programa -> Definicion*]]()

      for(Definicion d: Definicion)
            if(d instanceof DefVariable)
                  EJECUTAR [[d]]()
```

```
        <CALL MAIN>
        <HALT>

        for(Definicion d: Definicion)
                if(d instanceof DefFuncion)
                        EJECUTAR [[d]]()

EJECUTAR [[DefFuncion: Definicion -> Tipo Sentencia*]]()
        Definicion.Identificador <:>
        <ENTER> Definicion.NumeroBytesLocales

        for(Sentencia s:Sentencia*)
                if(!s instanceof DefVariable)
                        EJECUTAR[[s]]()

        if(Tipo.TipoRetorno instanceof TipoVoid)
                <RET> 0 <,> Definicion.numeroBytesLocales <,> Definicion.numeroBytesParam

EJECUTAR [[Escritura: Sentencia -> Exp]]()
        VALOR [[Exp]]()
        <OUT> Exp.Tipo.Sufijo()

EJECUTAR [[Lectura: Sentencia -> Exp]]()
        VALOR[[Exp]]()
        <IN> Exp.Tipo.Sufijo()
        <STORE> Exp.Tipo.Sufijo()

EJECUTAR [[Asignacion: Sentencia -> Exp1 Exp2]()
        DIRECCION[[Exp1]]()
        VALOR[[Exp2]]()
        cg.covertir(Exp2.Tipo,Exp1.Tipo)
        <STORE> Exp1.Tipo.Sufijo()

EJECUTAR [[SentenciaIf: Sentencia -> Exp if:Sentencia* else:Sentencia*]]()
        int etiqueta =  cg.getEtiquetas(2);
        VALOR[[Exp]]()
        <JZ><LABEL> etiqueta

        for(Sentencia s:if)
                EJECUTAR[[s]]()

        <JMP><LABEL> etiqueta+1
        <LABEL> etiqueta <:>

        for(Sentencia s:else)
                EJECUTAR [[s]]()

        <LABEL> etiqueta +1 <:>
```

```
EJECUTAR [[SentenciaWhile: Sentencia -> Exp Sentencia*]]()
	int etiqueta =  cg.getEtiquetas (2);
	<LABEL> etiqueta <:>
	VALOR[[Exp]]
	<JZ><LABEL> etiqueta+1

	for(Sentencia s:Sentencia*)
		EJECUTAR[[s]]()

	<JMP><LABEL> etiqueta
	<LABEL> etiqueta+1 <:>

EJECUTAR [[ InvocacionFuncionSent: Sentencia -> Identificador Exp*]]()
	for(Expresion e:Exp*)
		VALOR[[e]]()
	<CALL> Identificador.Nombre

	if(Identificador.Tipo.TipoRetorno != TipoVoid.getInstance())
		<POP> Identificador.Tipo.TipoRetorno.Sufijo();

EJECUTAR [[Return: Sentencia -> Exp]](DefFuncion)
	VALOR[[Exp]]()
	cg.convertir(Exp.Tipo, DefFuncion.Tipo.TipoRetorno);
	<RET> DefFuncion.TipoRetorno.numeroBytes
	<,> DefFuncion.numeroBytesLocales
	<,> DefFuncion.numeroBytesParam


VALOR [[LiteralEntero: Exp -> ConstanteEntera]]()
	<PUSHI> Exp.VALOR

VALOR [[LiteralCaracter: Exp -> ConstanteCaracter]]()
	<PUSHB> Exp.VALOR

VALOR [[LiteralReal: Exp -> ConstanteReal]]()
	<PUSHF> Exp.VALOR

VALOR [[Identificador: Exp -> ID]]()
	DIRECCION[[EXP]]()
	<LOAD> Exp.Tipo.Sufijo()

VALOR [[Aritmetica: Exp1 -> Exp2 Exp3 ]]()
	VALOR[[Exp2]]()
	cg.convertir(Exp2.Tipo,Exp1.Tipo)
	VALOR [[Exp3]]()
	cg.convertir(Exp3.Tipo,Exp1.Tipo)
	cg.aritmetica(Exp1.operador,Exp1.Tipo)

VALOR [[Comparacion: Exp1 -> Exp2 Exp3 ]]()
```

```
        superTipo = Exp2.Tipo.SuperTipo(Exp3.Tipo)
        VALOR[[Exp2]]()
        cg.convertir(Exp2.Tipo,superTipo)
        VALOR[[Exp3]]()
        cg.convertir(Exp3.Tipo,superTipo)
        cg.convertir(Exp1.operador,superTipo)

VALOR [[Cast: Exp1 -> TipoCast Exp2]]()
        VALOR[[Exp2]]()
        cg.cast(Exp2.Tipo, TipoCast)

VALOR [[Logica: Exp1 -> Exp2 Exp3 ]]()
        VALOR[[Exp2]]()
        VALOR[[Exp3]]()
        cg.logica(Exp1.operador)

VALOR [[Negacion: Exp1 -> Exp2]]()
        VALOR[[Exp2]]()
        <NOT>

VALOR [[AccesoCampo: Exp1 -> Exp2 ID]]()
        DIRECCION[[Exp1]]()
        <LOAD>Exp1.Tipo.Sufijo()

VALOR [[AccesoArray: Exp1 -> Exp2 Exp3 ]]()
        DIRECCION[[EXP1]]()
        <LOAD>Exp1.Tiop.Sufijo()

VALOR [[InvocacionFuncionExp: Exp -> Identificador Exp*]]()
        for(Expresion e:Exp*)
                VALOR[[e]]()
        <CALL> Identificador.Nombre


DIRECCION [[Identificador: Exp -> ID]]()
        if(Exp.Definicion.ambito == 0)
                <PUSHA> Exp.Definicion.Offset
        else
                <PUSH BP>
                <PUSHI> Exp.Definicion.Offset
                <ADDI>

DIRECCION [[ AccesoArray: Exp1 -> Exp2 Exp3 ]]()
        DIRECCION[[Exp2]]()
        VALOR[[Exp3]]()
        <PUSH> Exp1.Tipo.NumeroBytes
        <MUL>
        <ADD>
```

```
DIRECCION [[AccesoCampo: Exp1 -> Exp2 ID]]()
      DIRECCION[[Exp2]]
      <PUSH>Exp2.Tipo.get(ID).Offset
      <ADD>
```

Ampliaciones

Promoción implícita de tipos en:

- Comparación
- Aritmética
- Retorno de una función
- Invocación función como expresión (Paso de parámetros)
- Invocación función como sentencia (Paso de parámetros)
- Asignación

Operadores ++, -- para tipo float y entero.

Operadores +=, -=, *=, /= para tipo float, entero y carácter.