**Gramática para Sintáctico**

Los no terminales son los tokens escritos sin <> o minuscula

**--------------------------------------------------------------------**

**Versión 1.0**

**Algunos Terminales:**

Numero /\* CONSUMO UN TOKEN ENTERO\*/

Booleano -> true|false /\*CONSUMO UN TOKEN BOOLEANO \*/

Id -> /\* CONSUMO UN TOKEN DE TIPO ID \*/

Cadena -> ‘algo’ /\* CONSUMO UN TOKEN DE TIPO CADENA \*/

Tipo -> integer |boolean

Operador\_Booleano -> = | <> 🡪Token

Operador\_Booleano\_E -> < | > | <= | >=

Suma\_Resta -> + | - 🡪Token

Mult\_Div -> \* | / 🡪TOKEN

By -> // para el byval o byref

Show -> //aplica para show y showln

**Principal:**

**INICIAL SRL**

S’ -> <S>

INICIAL S -> <GLOBALES><BLOQUE>

GLOBALES -> <DECGL>|

λ

BLOQUE -> <FP><BLOQUE> |

λ

FP -> function Id(<PARAM>): Tipo; <BLOQUE1> end-func <EXP>; |

Procedure Id(<PARAM>); <BLOQUE1> end-proc;

**Bloques:**

BLOQUE1 -> <DECL> begin <BLOQUE2> |

λ

BLOQUE2 -> <LINEA>;<BLOQUE2> |

λ

LINEA -> Id<ASIG> |

if<EXP>then<BLOQUE2><BLOQUESI>end-if|

while<EXP> do <BLOQUE2> end-while|

<FUNC> |

ASIG -> (<PASAJE>)|

:= <EXP> |

[<EXP>] := <EXP>

BLOQUESI -> else <BLOQUE2> |

λ

FUNC -> read Id<IDAUX> |

Show <MOSTRAR>

**Parametrizaciónes:**

PARAM -> <TIPOPARAM> Id : Tipo <PARAM1> |

λ

PARAM1 -> ,<TIPOPARAM> Id : Tipo <PARAM1>|

λ

TIPOPARAM -> By |

λ

**Declaraciones:**

DECGL -> <VARG><DECGL> |

<CONSTS><DECGL> |

λ

DECL -> <VARS><DECL> |

<CONSTS><DECL> |

λ

**Constantes y variables:**

CONSTS -> const Id : <CONST1>;

CONST1 -> Tipo = NUMERO <CONST2>

| Tipo = BOOLEANO <CONST2>

CONST2 -> , Id : <CONST1> |

λ

VARS -> var Id <VAR1>;

VAR1 -> , Id <VAR1> |

: Tipo

Globales ----------------------------------------

VARG -> var Id <VARG1>;

VARG1 -> ,Id <VARG1> |

: Tipo |

[<Numero>] <VARG1>

**Funciones embebidas:**

MOSTRAR -> Cadena <MOSTRAR1> |

<EXP> <MOSTRAR1>

MOSTRAR1 -> , <MOSTRAR> | λ

IDAUX -> [EXP] | λ

**Expresiones:**

<EXP> -> <EXP> or <TERMBOOL> |

<TERMBOOL>

TERMBOOL -> <TERMBOOL> and <MINITERMBOOL> |

<MINITERMBOOL>

MINITERMBOOL -> not <MINITERMBOOL>|

<FACTBOOL>

FACTBOOL -> <EXPBOOL> |

<EXP2>

EXP2 -> <EXP2> Suma\_Resta <TERM>

| <TERM>

TERM -> <TERM> Mult\_Div <FACT>

| <FACT>

FACT -> (EXP)|

Id <ID1>|

Numero |

Booleano

EXPBOOL -> <EXP2> Operador\_Booleano <EXP2> | //tipos cualquieras, A = true o A = 2

<EXP2> Operador\_Booleano\_E <EXP2> //requiere enteros

ID1 -> [EXP] | (PASAJE) | λ

PASAJE -> <EXP> <PASAJE1> |

λ

PASAJE1 -> , <PASAJE> |

λ

**RESTRICCIONES DEL LENGUAJE**:

La función “mostrar” solo pude mostrar variables, constantes o vectores.

La función “leer” solo puede leer un número o booleano por vez invocada.

No se permite la doble negación not not A seria al pedo. Si se permite not(not A)

En EXPBOOL hay que chequear que ambos lados sean enteros.