Gramática para Sintáctico

==========

Versión 6.0

Principal: **MATIAS P.**

INICIAL	S'	->	S EOF
	S	->	GLOBALES SUBRUTINA
	GLOBALES	->	globales DECGL fin-globales; | λ
	SUBRUTINA	->	FP SUBRUTINA | λ
	FP	->	FUNCION | PROCEDIMIENTO

Funciones y procedimientos: **MATIAS P.**

FUNCION	->	ENCABEZADOF FUNCION'
ENCABEZADOF	->	funcion PALABRA(PARAMS): TIPO;
FUNCION'	->	adelantado; | DECL BLOQUEF
PROCEDIMIENTO	->	ENCABEZADOP PROCEDIMIENTO'
ENCABEZADOP	->	procedimiento PALABRA(PARAMS);
PROCEDIMIENTO'	->	adelantado; | DECL BLOQUEP

Bloques: **MATIAS D y GUILLE**

```
BLOQUEF          ->   comenzar BLOQUE fin-func EXP;

BLOQUEP          ->   comenzar BLOQUE fin-proc;

BLOQUE           ->   LINEA BLOQUE |
                      λ

LINEA            ->   BLOQUESI |
                      BLOQUEM |
                      PALRES |
                      PALABRA FPOASIG

FPOASIG          ->    (PASAJE); |
                      := EXP; |
                      [EXP] := EXP;

BLOQUEM          ->   mientras EXPBLOQUE hacer BLOQUE
                      fin-mientras;

BLOQUESI         ->   si EXPBLOQUE entonces BLOQUE BLOQUESI'

EXPBLOQUE        ->    EXPBOOL |
                       ESPAR |
                       ESIMPAR

BLOQUESI'        ->   fin-si; |
                      sino BLOQUE fin-si;

PALRES           ->   LEER |
                      MOSTRAR |
                      MOSTRARLN
```

Parametrizaciónes: **BRUNO**

```
PARAMS          -> TIPOPARAM PALABRA: TIPO PARAMS' |
                   λ

PARAMS'         -> ,TIPOPARAM PALABRA: TIPO PARAMS'|
                   λ

TIPOPARAM       -> ref |
                   val |
                   λ
```

Declaraciones: **NICO**

```
DECGL           ->   DECGLOBAL DECGL |
                     λ

DECGLOBAL       ->   VARSG |
                     CONSTS

DECL            ->   DECLARACIONES DECL |
                     λ

DECLARACIONES   ->   VARS |
                     CONSTS |
                     FP
```

Constantes y variables: **NICO**

```
CONSTS          ->   const CONSTS'

CONSTS'         ->   PALABRA TCONSTS

TCONSTS         ->   :TIPO = NUMERO CONSTS'' |
                     , CONSTS'

CONSTS''        ->   , CONSTS' |
                     ;

VARS            ->   var VARS'

VARS'           ->   PALABRA TVAR

TVAR            ->   , VARS' |
                     : TIPO INI VARS''

VARS''          ->   , VARS' |
                     ;
-- MATIAS P.
VARSG           ->   var VARSG'

VARSG'          ->   PALABRA TVARG

TVARG           ->   , VARSG' |
                     : TIPO INI VARSG'' |
                     [ NATURAL ] VECT VARSG''

VARSG''         ->   , VARSG' |
                     ;

VECT            ->   : TIPO |
                     λ
-- FIN
```

```
INI             ->   = NUMERO |
                     λ

NUMERO          ->   ENTERO |
                     NATURAL

TIPO            ->   entero |natural
```

Funciones requeridas: **BRUNO**

```
AENTERO         ->   aentero ( EXP )

ANATURAL        ->   anatural ( EXP )

ESPAR           ->   par ( EXP )

ESIMPAR         ->   impar ( EXP )

MOSTRARLN       ->   mostrarln MOSTRARAUX

MOSTRAR         ->   mostrar MOSTRARAUX

MOSTRARAUX      ->   CADENA MOSTRARAUX' |
                     PALABRA PAUX MOSTRARAUX'

MOSTRARAUX'     ->   , MOSTRARAUX |
                                 ;
PAUX            ->    [ EXP ] | λ

LEER            ->   leer PALABRA PAUX';

PAUX'           ->    [ NATURAL ] | λ
```

Terminales: **MATIAS P.**

```
ENTERO          ->   /* CONSUMO UN TOKEN DE TIPO ENTERO */
NATURAL         ->   /* CONSUMO UN TOKEN DE TIPO NATURAL */
PALABRA         ->   /* CONSUMO UN TOKEN DE TIPO PALABRA */
CADENA          ->   /* CONSUMO UN TOKEN DE TIPO CADENA */
```

Expresiones: **MATIAS P.**

```
EXPBOOL         ->    EXP EXPBOOL' |
                      AND |
                      OR

EXPBOOL'        ->    = EXP |
                      < EXP |
                      != EXP |
                      == EXP |
                      << EXP |
                      !== EXP

AND             ->    and(EXPBOOL,EXPBOOL)

OR              ->    or(EXPBOOL,EXPBOOL)


EXP             ->    TERM EXP'

EXP'            ->    + TERM EXP' |
                      - TERM EXP' |
                      ++ TERM EXP' |
                      -- TERM EXP' | λ

TERM            ->    FACT TERM'

TERM'           ->    * FACT TERM' |
                      / FACT TERM' |
                      ** FACT TERM' |
                      // FACT TERM'| λ

FACT            ->    (EXP) |
                      PALABRA FACT' |
                      NUMERO |
                      AENTERO |
                      ANATURAL

FACT'           ->    [EXP] | (PASAJE) | λ
```

```
PASAJE          ->   PALABRA PASAJE' |
                     NUMERO PASAJE' |
                     λ

PASAJE'         ->   ,PASAJE'' |
                     λ

PASAJE''        ->   PALABRA PASAJE'|
                     NUMERO PASAJE'
```

REFERENCIAS:

LISTO

EN TRABAJO

LISTO PERO PENDIENTE DE OTRA PRODUCCIÓN NO TERMINADO

RESTRICCIONES DEL LENGUAJE:

```
/*
La función "mostrar" solo pude mostrar variables, constantes o
vectores.
La función "leer" solo puede leer un número por vez invocada.

En el semántico hay que validar que no pueden haber dos expresiones
juntas que produzcan un número, ej: 1 1    1 2
*/
```

TO DO:

*GENRAR JAR

*GENERAR README

*COMENTAR SYSOUT

*HACER QUE EL JAR EJECUTABLE RECIBA EL ARCHIVO A ANALIZAR

*PONER CASOS DE PRUEBA

*EL DOCUMENTACION PASO A PASO DE COMO EJECUTAR EL PROGRAMA