Procesadores de Lenguajes.

# Introducción.

El lenguaje que vamos a trabajar está inspirado en el ***Lenguaje C***, de modo que asimos las reglas sintácticas usadas por tal lenguaje como base para el nuestro.

Las *palabras* *reservadas* serán en ***castellano***. Y la estructura de datos considerada como esencial será la ***pila LIFO*** *(last in first out)*; con las operaciones de manejo de datos de tipo pila.

Los elementos por los que se podrán componer nuestro nuevo tipo de dato elemental serán: *enteros, reales, caracteres o booleanos.*

Los subprogramas a realizar serán ***funciones*** y la estructura de control complementaria serán los ***switch/case***.

* Nuestro tipo de dato podrá realizar operaciones como: *insertar, sacar, conocer el tope, y saber si una pila está vacía.*
* También se podrán realizar operaciones aritméticas como: *producto, suma, resta y división.*

# Descripción.

A continuación se muestra la descripción formal de la sintaxis usando ***BFN***:

<programas> ::= <main><bloque>

<main> ::= **principal**

<bloque> ::= <inicio\_de\_bloque>

<declar\_de\_variables\_locales>

<declar\_de\_subprog>

<sentencias>

<fin\_de\_bloque>

<tipo\_dato> ::= **entero**

| **real**

| **booleano**

| **carácter**

| **pila entero**

| **pila real**

| **pila booleano**

| **pila carácter**

<variable> ::= <variable>

| <letra><numero>

| <letra>

<argumentos> ::= <tipo\_dato> <variable>

| <argumentos>

<inicio\_de\_bloque> ::= **{**

<declar\_de\_variables\_locales> ::= <marca\_ini\_var\_local>

<variables\_locales>

<marca\_fin\_var\_local>

<declar\_de\_subprogs> ::= <declar\_de\_subprogs><declar\_subprog>

<sentencias> ::= <sentencias><sentencia>

| <sentencia>

<sentencias> ::= <bloque>

| <sentencia\_asignacion>

| <sentencia\_if>

| <sentencia\_while>

| <sentencia\_entrada>

| <sentencia\_salida>

| <sentencia\_return>

| <sentencia\_switch>

<fin\_de\_bloque> ::= **}**

<letras> ::= <letras>

| **[a...z]**

| **[A...Z]**

<numero> ::= <numero>

| **[0...9]**

<marca\_ini\_var\_local> ::= **INI\_VAR\_LOCAL**

<variables\_locales> ::= <variables\_locales><cuerpo\_declar\_variables>

| <cuerpo\_declar\_variables>

<marca\_fin\_var\_local> ::= **FIN\_VAR\_LOCAL**

<declar\_subprog> ::= <cabecera\_subprog><bloque>

<sentencia\_asignacion>::= <variable> **=** <expresion>**;**

<sentencia\_if> ::= **si(**<condición>**)**<sentencia>

| **si(**<condición>**)**<sentencia> **sino** <sentencia>

<sentencia\_while> ::= **mientras(**<condición>**)**<sentencia>

<sentencia\_entrada> ::= <nombre\_entrada><lista\_variables>

<nombre\_entrada> ::= **entrada**

<lista\_variables> ::= <lista\_variables>**,** <variable>

| <variable>

<sentencia\_salida> ::= <nombre\_salida><lista\_expresiones\_o\_salidas>

<nombre\_salida> ::= **salida**

<lista\_expresiones\_o\_salida> ::=

<lista\_expresiones\_o\_salidas>, <expresion\_cadena>

| <expresion\_cadena>

<expresion\_cadena> ::= <expresion>

| <cadena>

<sentencia\_return> ::= **devolver**<expresion>;

<sentencia\_switch> ::= **interruptor(**<variable>**){**<bloque\_switch>**}**

<cuerpo\_declar\_variables> ::= <tipo\_dato><lista\_variables>;

<condición> ::= <variable><binario><variable>

| <op\_unario><variable>

<expresiones\_return> ::= <variable>

| **0**

| **1**

<bloque\_switch> ::= **caso**<expresion>:<sentencia>**detener**;<bloque\_switch>

<cabecera\_subprog> ::= <tipo\_dato><variable>**(**<param>**)**

<param> ::= <param>**,** <tipo\_dato> <variable>

| <tipo\_dato><variable>

<expresión> ::= **(**<expresión>**)**

| <op\_unario><expresión>

| <expresión><binario><expresión>

| <variable>

| <constante>

| <función>

<op\_unario> ::= **&** | **#** | **!** | **+** | **-**

<binario> ::= **++** | **+** | **-** | **\*** | **/** | **%** | **&&** | **||** | **&** | **|** | **>>** | **<<**

<constante> ::= <const\_entero>

| <const\_real>

| <const\_booleano>

| <const\_caracter>

| <const\_pila>

<const\_entero> ::= <numero>

<const\_real> ::= <numero> . <numero>

<const\_booleano> ::= **Verdadero** | **Falso**

<const\_caracter> ::= **' [a...z] '**

| **'[A...Z]'**

<cadena> ::= “<letra>”

| <espacio>

| <cadena>

<const\_pila> ::= <const\_pila\_entero>

| <const\_pila\_real>

| <const\_pila\_booleano>

| <const\_pila\_caracter>

<const\_pila\_entero> ::= <inicio\_const\_pila> <lista\_const\_entero>

<final\_const\_pila>

<lista\_const\_entero> ::= <lista\_const\_entero>**,** <const\_entero>

| <const\_entero>

<const\_pila\_real> ::= <inicio\_const\_pila><lista\_const\_real><final\_const\_pila>

<inicio\_const\_pila> ::= **[**

<final\_const\_pila> ::= **]**

<lista\_const\_real> ::= <lista\_const\_real>**,** <const\_real>

| <const\_real>

<const\_pila\_booleano> ::= <inicio\_const\_pila>< lista\_const\_booleano>

<final\_const\_pila>

<lista\_const\_booleano> ::= <lista\_const\_boolenao><const\_booleano>

| <const\_booleano>

<lista\_const\_caracter> ::= <inicio\_const\_pila> <lista\_const\_ caracter>

<final\_const\_pila> <final\_const\_ caracter>

<lista\_const\_caracter>::= <lista\_const\_caracter><const\_caracter>

| <const\_caracter>

<función> ::= <variable>**(**<lista\_expresion>**)**

<lista\_expresion> ::= <lista\_expresion>**,** <expresion>

| <expresion>

# Semántica.

Las instrucciones en nuestro lenguaje son:

1. **while**: Se utiliza para realizar una serie de iteraciones sobre un bloque de instrucciones hasta que se cumple una condición. While(<condición>){…instrucciones…}
2. **switch/case**: Se utiliza para, dado un parámetro, cambiar el flujo del programa. Switch(<variable>){case 1: ….instrucciones… break; case 2:…}. Y tantos case como se quieran.
3. **pila++elemento**: Inserta un elemento en el tope de la pila.
4. **&pila**: Borra el tope de la pila.
5. **!pila:** Para saber si una pila está vacía o no.
6. **Pila\*x:** Se realiza el producto con tres posibilidades para ‘x’, puede ser:
   1. **Pila:** realiza el producto de las pilas elemento a elemento.
   2. **Numero:** realiza el producto de todos los elementos de la pila por el número ‘x’.
   3. **X\*pila:** Igual que el caso 6-b.
7. **Pila+x:** Se realiza la suma como en el caso 6-a, b y c.
8. **Pila-x**: Se realiza la diferencia como en el caso 6-a, b y c.
9. **División:** Se realiza la división como en el caso 6-a, b y c.

Cuando se realiza una operación aritmética a dos pilas, ambas deben ser del mismo tipo y deben tener el mismo número de elementos. Cuando se opera con un dato y una pila también deben compartir el mismo tipo básico.

Cuando se realiza una asignación entre pilas, la sentencia es válida sí y solo sí ambas tienen el mismo tipo.

# Componentes sintácticos.

|  |  |  |
| --- | --- | --- |
| **Token** | **Código** | **Patrón** |
| <main> |  | principal |
| <binario> |  | ++ | \* | / | % | && | || | == | | | << | >> | < | > | != | >= | <= |
| <mas\_menos> |  | + | - |
| <numero> |  | (0 | 1 | 2 | … | 9)+ |
| <op\_unario> |  | & | # | ! |
| CASE |  | caso |
| PARIZQ |  | ( |
| PARDER |  | ) |
| ASIG |  | = |
| IF |  | si |
| ELSE |  | sino |
| WHILE |  | mientras |
| <nombre\_entrada> |  | entrada |
| <nombre\_salida> |  | salida |
| <marca\_fin\_var\_local> |  | FIN\_VAR\_LOCAL |
| <marca\_ini\_var\_local> |  | INI\_VAR\_LOCAL |
| <fin\_de\_bloque> |  | } |
| <inicio\_de\_bloque> |  | { |
| <tipo\_dato> |  | entero | real | booleano | carácter | pila entero | pila real | pila booleano | pila carácter |
| RETURN |  | devolver |
| SWITCH |  | interruptor |
| <const\_booleano> |  | Verdadero | Falso |
| <inicio\_const\_pila> |  | [ |
| <final\_const\_pila> |  | ] |
| FIN\_SENTENCIA |  | ; |
| <cadena> |  | \”[^\”]\*” |
| <real> |  | <numero>.<numero> |
| <nombre> |  | [A-Za-z\_]([A-Za-z0-9\_])\* |
| <caracter> |  | \’[^\’]\’ |
| <dos\_puntos> |  | : |
| BREAK |  | detener |

# Gramática abstracta

La gramática abstracta eliminando las producciones en las que derivan los tokens.

<programas> ::= **MAIN**<bloque>

<bloque> ::= **INI\_BLOQUE**

<declar\_de\_variables\_locales>

<declar\_de\_subprog>

<sentencias>

**FIN\_BLOQUE**

<variable> ::= **NOMBRE**

<argumentos> ::= **TIPO\_DATO** <variable>

| <argumentos>

<declar\_de\_variables\_locales> ::= **I\_VAR\_LOCAL**

<variables\_locales>

**F\_VAR\_LOCAL**

<declar\_de\_subprogs> ::= <declar\_de\_subprogs><declar\_subprog>

<sentencias> ::= <sentencias><sentencia>

| <sentencia>

<sentencias> ::= <bloque>

| <sentencia\_asignacion>

| <sentencia\_if>

| <sentencia\_while>

| <sentencia\_entrada>

| <sentencia\_salida>

| <sentencia\_return>

| <sentencia\_switch>

<variables\_locales> ::= <variables\_locales><cuerpo\_declar\_variables>

| <cuerpo\_declar\_variables>

<declar\_subprog> ::= <cabecera\_subprog><bloque>

<sentencia\_asignacion>::= <variable> **ASIG** <expresion>**FIN\_SENTENC**

<sentencia\_if> ::= **IF PARIZQ**<condición>**PARDER**<sentencia>

| **IF PARIZQ**<condición>**PARDER**<sentencia> **ELSE** <sentencia>

<sentencia\_while> ::= **WHILE PARIZQ** <condición> **PARDER** <sentencia>

<sentencia\_entrada> ::= **INPUT** <lista\_variables>

<lista\_variables> ::= <lista\_variables> **BINARIO** <variable>

| <variable>

<sentencia\_salida> ::= **OUTPUT** <lista\_expresiones\_o\_salidas>

<lista\_expresiones\_o\_salida> ::=

<lista\_expresiones\_o\_salidas> **BINARIO** <expresion\_cadena>

| <expresion\_cadena>

<expresion\_cadena> ::= <expresion>

| **CADENA**

<sentencia\_return> ::= **RETURN** <expresion> **FIN\_SENTENC**

<sentencia\_switch> ::=

**SWITCH PARIZQ** <variable> **PARDER INI\_BLOQUE** <bloque\_switch> **FIN\_BLOQUE**

<cuerpo\_declar\_variables> ::= **TIPO\_DATO** <lista\_variables> **FIN\_SENTENC**

<condición> ::= <variable> **BINARIO** <variable>

| **BINARIO** <variable>

<expresiones\_return> ::= <variable>

| **CONS\_BOOL**

<bloque\_switch> ::=

**CASE** <expresion> **DOS\_PUNTOS** <sentencia> **BREAK** **FIN\_SENTENC**<bloque\_switch>

<cabecera\_subprog> ::= **TIPO\_DATO** <variable> **PARIZQ** <param> **PARDER**

<param> ::= <param> **BINARIO TIPO\_DATO** <variable>

| <tipo\_dato><variable>

<expresión> ::= **PARIZQ** <expresión> **PARDER**

| **UNARIO** <expresión>

| <expresión> **BINARIO** <expresión>

| <variable>

| <constante>

| <función>

<constante> ::= **ENTERO**

| **REAL**

| **BOOL**

| **CHAR**

| <const\_pila>

<const\_pila> ::= <const\_pila\_entero>

| <const\_pila\_real>

| <const\_pila\_booleano>

| <const\_pila\_caracter>

<const\_pila\_entero> ::= **I\_CONS\_PILA** <lista\_const\_entero>

**F\_CONS\_PILA**

<lista\_const\_entero> ::= <lista\_const\_entero> **BINARIO ENTERO**

| **ENTERO**

<const\_pila\_real> ::= **I\_CONS\_PILA** <lista\_const\_real> **F\_CONS\_PILA**

<lista\_const\_real> ::= <lista\_const\_real> **BINARIO** <const\_real>

| <const\_real>

<const\_pila\_booleano> ::= **I\_CONS\_PILA** <lista\_const\_booleano>

**F\_CONS\_PILA**

<lista\_const\_booleano> ::= <lista\_const\_boolenao>

| **BOOL**

<lista\_const\_caracter> ::= **I\_CONS\_PILA** <lista\_const\_ caracter>

**F\_CONS\_PILA** <final\_const\_ caracter>

<lista\_const\_caracter>::= <lista\_const\_caracter> **CHAR**

| **CHAR**

<función> ::= <variable> **PARIZQ** <lista\_expresion> **PARDER**

<lista\_expresion> ::= <lista\_expresion> **BINARIO** <expresion>

| <expresion>