CAPÍTULO 2: ANÁLISIS DEL PSEUDOLENGUAJE

2.1.Principales características del pseudolenguaje.

El pseudolenguaje empleado en este proyecto tiene las siguientes características:

- Perteneciente al paradigma de programación imperativa y con características similares a otros lenguajes de este paradigma, como PASCAL o MODULA – 2.
- Permite escribir algoritmos flexibles y legibles, lo que contrasta con otros lenguajes que son más crípticos y difíciles de leer, como por ejemplo C.
- Declaración obligatoria de las variables, tipos estructurados y constantes que se van a utilizar. Para ello se utiliza la zona de declaración correspondiente.
- Los algoritmos escritos en este lenguaje tienen una apariencia similar al lenguaje natural, cosa que se facilita con el empleo de palabras reservadas y funciones de entrada y salida en castellano.
- Las palabras reservadas deben aparecer en mayúsculas (por ejemplo: SI, ENTONCES, FINSI, etc.) y no pueden utilizarse como identificadores.
- Se distingue entre mayúsculas y minúsculas. Así, por ejemplo CONT, cont, Cont y CoNt son identificadores distintos.
- A diferencia de la mayoría de lenguajes, las sentencias como asignaciones y llamadas a funciones no se acaban con punto y coma.
- Los espacios en blanco, saltos de línea y tabuladores entre tokens son ignorados y no tienen efecto en los algoritmos.
- Todos los bloques de sentencias (como funciones, estructuras de control, etc.) están delimitados por palabras reservadas que indican el comienzo del bloque y el final del mismo.
- Es un lenguaje procedural, es decir, permite la definición de procedimientos y funciones que facilitan la programación descendente y división de las tareas a realizar.

2.2. Estructura de un algoritmo en pseudolenguaje

Cada algoritmo escrito en pseudolenguaje comienza con la palabra reservada ALGORITMO, seguida por un identificador que indica su nombre.

A continuación deben declararse todas las constantes, los tipos y las variables que vayan a emplear en el algoritmo. Para ello se utilizan las zonas de declaración que comienzan con las palabras reservadas CONSTANTES, TIPOS y VARIABLES respectivamente.

Ninguna de estas zonas es obligatoria pero si debe respetarse su orden, es decir, el traductor no aceptará que se declaren las variables o los tipos antes que las constantes.

Las constantes deben declararse mediante un identificador, el símbolo "<-" y el valor constante que toma. Por ejemplo:

```
Pi <- 3.14159
```

Los tipos se declaran mediante un identificador, el símbolo igual ("=") y la definición del tipo. Por ejemplo :

```
Tedad =  [0 .. 120]
```

Y las variables se declaran mediante uno o más identificadores separados por comas que comparten el tipo, a continuación dos puntos (":"), y por último el nombre del tipo de las variables. Por ejemplo:

```
contador, inicio: ENTERO
```

A continuación se deben declarar todas las funciones y procedimientos que vayan a emplearse. Se pueden declarar tantas funciones y procedimientos como se deseen y en cualquier orden. La palabra reservada que indica que comienza la declaración de una función es FUNC y la que

indica el comienzo de un procedimiento es PROC. A continuación de la palabra reservada, al igual que en el caso del algoritmo, debe indicarse el nombre de la función o el procedimiento. Tras esto deben aparecer los parámetros entre paréntesis separados por punto y coma. Tras el paréntesis derecho, en el caso de las funciones se debe indicar el tipo que se devuelve.
Dentro de una función o procedimiento pueden declararse constantes, tipo y variables locales, pero no podrán declararse funciones o procedimientos anidados, por razones que se explicarán en el capítulo siguiente. Tras las declaraciones deben aparecer las sentencias del procedimiento o la función que se inicia con la palabra reservada INICIO y acaba con la palabra reservada FIN.

Por último, aparece la zona de sentencias del programa principal, delimitada por las palabras reservadas INICIO y FIN.

Así, la estructura de un algoritmo escrito en pseudolenguaje puede representarse de la siguiente forma:

```
ALGORITMO nombre_algoritmo
    CONSTANTES
        declaración_constantes
    TIPOS
        declaración_tipos
    VARIABLES
        declaración_variables
    PROC nombre_procedimiento ( parámetros_procedimiento )
        declaraciones_procedimiento
        INICIO
            sentencias_procedimiento
        FIN

    FUNC nombre_función ( parámetros_función ): tipo_devuelto
        declaraciones_función
        INICIO
            sentencias_función
        FIN
 ...
INICIO
sentencias_algoritmo
FIN
```

2.3. Tipos básicos del pseudolenguaje.

Los tipos básicos que pueden emplearse en el pseudolenguaje son los siguientes:

- *ENTERO*: Comprende los números enteros en un rango de [- *MAXENTERO,*0] a [0,*MAXENTERO*], donde *MAXENTERO* es el máximo valor entero permitido por el tipo. En la práctica, y tras la traducción, *MAXENTERO* depende del tipo *int* de C++.

- *NATURAL:* Comprende los números enteros positivos y el cero. Abarca por tanto el rango [0, *MAXNATURAL*].
- *LÓGICO:* Los valores permitidos en este tipo son las constantes lógicas CIERTO y FALSO, y sobre los valores de este tipo se pueden emplear los distintos operadores lógicos.
- *CARACTER:* Comprende los distintos caracteres alfanuméricos que define el código ASCII.
- *REAL:* Comprende números reales representados internamente en el formato de punto flotante. La parte entera debe separarse de la decimal mediante un punto. La precisión y magnitud de este tipo depende debido a la traducción del tipo *float* de C++.
- *Tipos Enumerados:* El usuario puede definir tipos enumerados mediante la enumeración del conjunto de identificadores que forman los valores del nuevo tipo. Estos identificadores se utilizan entonces como constantes en los programas, lo que proporciona mayor claridad. Para declarar uno de estos tipos se deben encerrar los valores entre paréntesis y separar por comas. Por ejemplo:

Tcolor = (ROJO, AZUL, AMARILLO)

- *Tipos Subrango:* Permite declarar variables que estén entre dos límites, un máximo y un mínimo que se especifican. Los tipos subrangos solo pueden emplearse con tipos ordinales (todos los tipos básicos anteriores menos el tipo REAL). Para declarar uno de estos tipos se deben separar los dos valores extremos por dos puntos (“..”) y encerrar entre corchetes (“[“ y “]”).Por ejemplo: Si

Tcolor = (ROJO, AZUL, AMARILO, VERDE, NARANJA)

podemos definir el tipo subrango Tcolorbásico como

Tcolorbásico = [ROJO .. AMARILLO]

2.4. Tipos estructurados del pseudolenguaje.

A partir de los tipos básicos anteriores, se pueden formar otras estructuras como las siguientes:

- REGISTRO: Se trata de una colección fija de información relativa a un solo objeto. Nos podemos referir a esta información como un todo o partes de la misma. Cada una de estas partes que compone el registro se denomina campo. Para definir un tipo registro se debe comenzar con la palabra reservada REGISTRO. A continuación se incluyen todos los campos que forman el registro como si se tratase de variables ordinarias. Por último se concluye con la palabra reservada FINREGISTRO. Por ejemplo, para definir un registro que contenga los datos de un número complejo, podemos emplear la siguiente estructura:

```
TComplejo = REGISTRO
                parte_real, parte_imaginaria: REAL
            FINREGISTRO
```

  Cuando se ha definido una variable de este tipo podemos referirnos a uno de sus campos utilizando el operador punto (“.”). Así, si tenemos la declaración de la variable comp1:

```
VARIABLES
  comp1:Tcomplejo
```

  podemos acceder a su parte real escribiendo:

```
comp1.parte_real.<- 2.0
```

- ARRAY: Este tipo define un conjunto de elementos ordenados de tal forma que un conjunto de elementos de un tipo ordinal definen unívocamente la posición de cada elemento dentro del array. Los elementos que forman el array son todos del mismo tipo, que se denomina tipo base. El tipo ordinal que sirve para indexar cada uno de estos elementos se denomina tipo índice. El número de elementos del array es finito. Los arrays

pueden tener una o varias dimensiones, y para indexar un elemento del array se necesitarán tantos elementos de los tipos índices como dimensiones tenga el array. La declaración de un array se hace de la siguiente forma:

```
TipoArray = ARRAY TipoIndice1 TipoIndice2 ... TipoIndiceN DE TipoBase.
```

El tipo índice debe ser un tipo subrango, enumerado o carácter. En cambio, el tipo base puede ser cualquier tipo simple o estructurado.

Para acceder a los elementos individuales de un array se emplea el operador selector ("[ ]"), encerrando entre corchetes tantos como tipos índices sean necesarios según el número de dimensiones del array.

Por ejemplo, si tenemos la siguiente definición de array:

```
TIPOS
        Tmatriz = ARRAY [0..N-1] [0.. N-1] DE ENTERO
VARIABLES
        matriz: Tmatriz
```

Para acceder a un elemento de tipo entero de este array debemos escribir

```
matriz [i] [j]
```

donde i y j son valores de tipo entero o natural comprendidos entre 0 y N-1.

- PUNTERO: El tipo puntero sirve para definir variables cuyo contenido es una dirección de una parte de la memoria donde hay un dato que nos interesa. Para declarar un tipo puntero en nuestro pseudolenguaje se emplea la palabra reservada PUNTERO seguida por la palabra reservada A y luego el tipo de la variable anónima de memoria a la que apuntará el puntero. Por ejemplo, para definir un puntero a un entero, escribimos:

```
TpunteroAEntero = PUNTERO A ENTERO
```

Un valor especial que puede tomar una variable de tipo puntero es NULO, e indica que esa variable no apunta actualmente a ninguna dirección de memoria. Para acceder al contenido de la memoria apuntada por el puntero, se emplea el operador de dereferenciación ("^"). Por ejemplo si ptr es de tipo TpunteroAEntero, podemos acceder al valor entero que contenga mediante la expresión:

```
ptr^
```

Para crear una nueva variable anónima a la que pueda apuntar una variable de tipo puntero se emplea la función NUEVO, seguida entre paréntesis por el tipo de la variable anónima. Para liberar la memoria que ha sido asignada a un puntero se emplea la función ELIMINAR seguida por el puntero que apunta a la variable anónima encerrado entre paréntesis. Por ejemplo:

```
ptr <- NUEVO(ENTERO)
...
ELIMINAR(ptr).
```

- FICHERO: Los ficheros son conjuntos de información relacionada, tratados como unidades de almacenamiento en memoria secundaria y organizados de forma estructurada para facilitar la búsqueda de datos individuales. Están compuestos por registros homogéneos que contienen información organizada en campos, que son las unidades mínimas de procesamiento con significado propio. Para poder acceder a un fichero del pseudolenguaje se debe declarar una variable de un tipo especial, que se identifica por la palabra reservada FICHERO. Esta variable contiene un área de memoria para almacenar el registro actual y el cursor del fichero. Un ejemplo de declaración de variable de tipo fichero es el siguiente:

```
VARIABLES
        Mifichero: FICHERO
```

Para asociar un fichero físico existente con una variable de tipo FICHERO, o descriptor de fichero existen varias funciones. La función ABRIR asocia un fichero ya existente con un descriptor y sitúa el cursor al inicio del fichero. Esta función devuelve un descriptor de tipo fichero y debe pasársele como argumento una cadena de caracteres con el nombre del fichero. Por ejemplo:

```
fich1 <- ABRIR (“FICHERO1.DAT”)
```

También puede usarse la primitiva AÑADIR (que debe escribirse ANADIR), que en este caso sitúa el cursor al final del fichero. El fichero debe existir previamente. Por ejemplo:

```
fich1 <- ANADIR (“FICHERO1.DAT”)
```

La última manera existente de asociar un fichero a un descriptor es con la función CREAR, que crea y abre un fichero. Si el fichero ya existía, destruye su contenido:

```
fich1 <- CREAR (“FICHERO1.DAT”)
```

Una vez que se ha trabajado con un fichero, es conveniente liberar sus recursos, para lo que se usa la sentencia CERRAR, pasándole como argumento el descriptor del fichero. A continuación se muestra un ejemplo:

```
CERRAR (fich1)
```

Para trabajar con los ficheros en el pseudolenguaje, se pueden usar dos clases de primitivas de acceso: para ficheros binarios (que almacenan su contenido en el mismo formato que la memoria principal) o para ficheros de texto (que almacenan su contenido en forma de cadenas de caracteres).

A continuación se considerarán las primitivas de acceso a ficheros binarios. Para leer de estos ficheros, se utiliza la sentencia LEERBIN. A esta sentencia deben pasársele dos parámetros: el primero es el descriptor del fichero con el que se va a trabajar, y el segundo es una variable de cualquier tipo donde quedará almacenada la información que se lea del fichero. Por ejemplo:

```
fich1 <- ABRIR ("FICHERO1.DAT")
LEERBIN(fich1,dato)
```

Esta sentencia recogerá del fichero tantos bytes como sea el tamaño del tipo de dato. Estos bytes son transferidos a la variable dato, y se mueve el cursor del fichero al siguiente registro.

Para poder leer la información contenida en un fichero binario se utiliza la primitiva de acceso ESCRIBIRBIN, que tiene los mismos parámetros que LEERBIN, pero en este caso la información que contiene la variable se escribe en el fichero. Por ejemplo:

```
fich1 <- CREAR ("FICHERO1.DAT")
ESCRIBIRBIN(fich1,dato)
```

En este caso se transfieren tantos bytes al fichero como sean necesarios para contener la información de la variable dato y se mueve el cursor al siguiente hueco libre para escribir. También existe una función que sirve para verificar si se ha llegado al final del fichero en la última operación realizada sobre él (generalmente de lectura). Esta función se identifica con la palabra reservada EOF, y devuelve un valor de tipo LOGICO si le pasamos como parámetro el descriptor del fichero del que queremos comprobar si se ha alcanzado el fin. Por ejemplo:

```
MIENTRAS ¬ EOF (fich1) HACER
        LEERBIN(fich1,dato)
        ...
```

Esta función también puede utilizarse en el caso de los ficheros de texto.
Aunque las primitivas descritas se emplean para el acceso secuencial a los ficheros (los registros se escriben y leen de consecutivamente), también es posible trabajar con primitivas de acceso directo con los ficheros binarios. Por ejemplo, la primitiva BUSCAR sitúa el cursor del fichero en el número de byte del mismo que se le indique. Si esta posición es mayor que el número de bytes del fichero, entonces la función EOF devolverá CIERTO en la próxima llamada. El primer parámetro de BUSCAR es el descriptor del fichero y el segundo es un número natural que indica la posición que deseamos. Por ejemplo:

```
BUSCAR (fich1, 5* Tamano( ENTERO))
```

En este caso, si se trata de un fichero binario que contiene números enteros, situaríamos el cursor del fichero al inicio del sexto número escrito en el mismo. Dado que la dirección debe indicarse en bytes y no en número de registros del fichero, es de mucha ayuda la utilización de la función Tamaño (se debe escribir Tamano), que devuelve el número de bytes necesarios para contener un determinado tipo.

Otra primitiva de acceso secuencial a los ficheros es POSICION, que devuelve la posición actual (en número de bytes) del cursor del fichero. El parámetro que debe pasársele es el descriptor de fichero cuya posición queremos averiguar. Por ejemplo:

```
Escribir( "El cursor se encuentra en el registro número: ")
Escribir( POSICION ( fich1) / Tamano (ENTERO) )
```

Estas sentencias escribirían la posición actual del cursor en un fichero de números enteros.

En cuanto a los ficheros de texto, además de las distintas formas de apertura, el cierre y el control de fin de fichero, existen las siguientes primitivas de acceso:

ESCRIBIR tiene como primer parámetro el descriptor del fichero y como segundo parámetro una variable de un tipo simple o cadena de caracteres. Permite escribir el contenido de esa variable en un fichero en formato texto. Por ejemplo:

```
ESCRIBIR(fich1, dato)
```

Si el tipo de la variable dato no fuese una cadena de caracteres o un tipo simple, se produciría un error, ya que es necesaria esta condición para poder hacer la transformación al formato texto.

La primitiva para el leer el contenido de un fichero de tipo texto es LEER, cuyo primer parámetro es también el descriptor de fichero, y el segundo parámetro una variable de tipo simple o cadena donde se almacena la información que se lee del fichero. Un ejemplo de la utilización de esta sentencia es el siguiente:

```
MIENTRAS ¬ EOF (fich1) HACER
        LEER(fich1,dato)
        ...
```

Además de estas operaciones, también es posible renombrar los ficheros, borrarlos o comprobar su existencia. Para ello se utilizan las sentencias que se explican a continuación.

La sentencia RENOMBRAR sirve para cambiarle el nombre a un fichero. Para ello se utiliza la palabra reservada RENOMBRAR y dos parámetros de tipo cadena. El primero de ellos indica el nombre antiguo del fichero y el segundo el nuevo nombre que se le da. Por ejemplo, si queremos que un fichero pase de llamarse FICHERO1.DAT a llamarse FICHERO2.DAT tendremos que escribir:

```
RENOMBRAR("FICHERO1.DAT","FICHERO2.DAT")
```

Si deseamos eliminar un fichero, utilizaremos la sentencia BORRAR, a la que debe pasársele como parámetro el nombre del fichero a borrar. Por ejemplo:

```
BORRAR("FICHERO2.DAT")
```

También se puede comprobar si existe o no un fichero determinado mediante la función EXISTE, que devuelve un valor lógico. Este valor será CIERTO si el fichero existe y FALSO en otro caso. A esta función debe pasársele como parámetro el nombre del fichero cuya existencia queremos comprobar. Por ejemplo:

```
SI EXISTE ("FICHERO2.DAT") ENTONCES
        fich1 <- ABRIR("FICHERO2.DAT")
ENOTROCASO
        fich1 <- CREAR("FICHERO2.DAT")
FINSI
```

También es posible crear y borrar directorios utilizando directivas del pseudolenguaje. La directiva CREARDIR, seguida del nombre del directorio, permite crear un nuevo directorio. Por ejemplo:

```
CREARDIR("Nuevo")
```

Para borrar un directorio, se utiliza la sentencia BORRARDIR, seguida del nombre del directorio a borrar. Un ejemplo:

```
BORRARDIR ("Temp")
```

2.5. Expresiones en el pseudolenguaje.

Las expresiones en general se utilizan en los programas con el propósito de obtener un valor. Al igual que las variables, las expresiones tienen un tipo que puede ser REAL, ENTERO, LÓGICO, etc.

Las expresiones más simples de cada tipo son las constantes y variables de ese tipo. También se pueden conseguir expresiones más complejas combinando las variables y constantes con un conjunto de operadores que varían según el tipo de expresión.

A continuación se explican las características de las distintas expresiones en nuestro pseudolenguaje:

- *Expresiones enteras:* En el pseudolenguaje no se hace distinción entre expresiones enteras y naturales, por lo tanto en este apartado nos referimos a ambas. Los operadores utilizados en las expresiones enteras son:
    - *Operador suma (+)*
    - *Operador resta (-)*
    - *Operador multiplicación (*)*
    - *Operador división entera (/)*
    - *Operador módulo (MOD)*

  Todos los operadores anteriores son infijos, y la multiplicación, la división y el módulo tienen prioridad sobre la suma y la resta. Existe además un operador prefijo, que comparte símbolo con el operador resta ("-"), y que sirve para denotar números negativos.
- *Expresiones reales*: A diferencia del operador módulo (MOD), que no está definido para las expresiones reales, estas tienen la misma forma que las expresiones enteras, aunque su tipo es distinto.
- *Expresiones lógicas:* Además de las constantes lógicas CIERTO y FALSO, y las variables de tipo lógico, las expresiones lógicas pueden formarse con los siguientes operadores:
    - *Operador lógico AND (\/)*

- *Operador lógico OR (/\)*
- *Operador lógico NOT (¬)*

Los operadores AND y OR son de tipo infijo y el operador NOT de tipo prefijo. El operador AND tiene prioridad sobre el operador OR.

Además se pueden generar expresiones de tipo lógico a partir de expresiones de tipo entero o real. Para ello se emplean los operadores relacionales:

- *Operador igualdad (=)*
- *Operador de desigualdad (#)*
- *Operador mayor (>)*
- *Operador mayor o igual (>=)*
- *Operador menor(<)*
- *Operador menor o igual(<=)*

Los operadores relacionales son todos de tipo infijo y tienen mayor prioridad que el resto de los operadores lógicos.

- *Expresiones de caracteres:* Además de las variables de tipo carácter y las constantes de este tipo (caracteres encerrados entre comillas “”), las expresiones de tipo carácter pueden formarse con los siguientes operadores:
  - *Operador mayúscula (CAP)*
  - *Operador carácter (CHR)*

  El primero de los operadores se utiliza en forma de función, y devuelve la letra mayúscula correspondiente al carácter que se pasa como parámetro. Por ejemplo CAP(“a”) es “A”, y CAP(“A”) también es “A”. El segundo operador se utiliza de la misma forma, pero devuelve el carácter correspondiente a la posición del código ASCII del número entero que se pasa como parámetro.
- *Expresiones ordinales:* Debido a sus características comunes (como por ejemplo que todo valor tiene un sucesor y un predecesor), los tipos ordinales (entero, natural, carácter, subrango y enumerado) poseen una serie de operadores comunes que son los siguientes:
  - *Operador Ordinal (ORD)*
  - *Operador Sucesor (SUC)*
  - *Operador Predecesor (PRED)*

Todos estos operadores se utilizan de la misma forma que CAP y CHR. El operador ORD devuelve el número entero que indica la posición que ocupa dentro de su tipo el valor ordinal que se pasa como parámetro. El operador SUC devuelve el siguiente valor siguiendo el orden del tipo y PRED devuelve el valor anterior.

2.6. Sentencias del pseudolenguaje.

Las sentencias del pseudolenguaje se incluyen en los bloques delimitados por las palabras reservadas INICIO y FIN, tanto en el algoritmo principal como en las funciones y procedimientos.

Las sentencias permitidas en el pseudolenguaje son las siguientes:

- *Asignación*: Consiste en asignar a una variable el valor de una expresión del mismo tipo de la variable. El operador de asignación es "<-". Un ejemplo sencillo de asignación de una variable de tipo entero es:

```
cont <- cont + 1
```

- *Sentencia SI - ENTONCES - ENOTROCASO - FINSI*: Se trata de la clásica sentencia de selección simple. Entre las palabras reservadas SI y ENTONCES debe aparecer una expresión de tipo lógico que determina si se ejecutan las sentencias contenidas en el bloque ENTONCES o en el bloque ENOTROCASO. El bloque ENOTROCASO es opcional y cuando no aparece no se ejecuta ninguna opción en caso de que no se cumpla la condición de la sentencia. Un ejemplo del uso de esta sentencia es:

```
SI a > b ENTONCES
        mayor <- a
ENOTROCASO
        mayor <- b
FINSI
```

- *Sentencia MIENTRAS - HACER - FINMIENTRAS:* Es la sentencia fundamental para la construcción de bucles. Entre las palabras reservadas MIENTRAS y HACER debe aparecer una expresión de tipo lógico. Mientras el valor resultante de evaluar esta expresión sea CIERTO, se ejecutarán las sentencias comprendidas entre las palabras reservadas HACER Y FINMIENTRAS. Un ejemplo del uso de esta sentencia es:

```
MIENTRAS i < N HACER
        suma <- suma + vector[i]
        i <- i + 1
FINMIENTRAS
```

- *Sentencia PARA - HASTA - (PASO) – HACER- FINPARA:* Esta sentencia para ejecutar un conjunto de sentencias mientras se asigna a una variable de control una progresión de valores. Entre las palabras reservadas PARA y HASTA debe aparecer la asignación del valor inicial a la variable de control. Tras la palabra reservada HASTA aparece el último valor a tomara por la variable de control. Entre paréntesis y de forma opcional puede aparecer la palabra reservada PASO seguida del valor por el que debe incrementarse o decrementarse la variable de control en cada paso. Si no se indica de forma explícita el valor por defecto es 1. Por último entre las palabras HACER y FINPARA se colocan las sentencias que se ejecutan en cada iteración. Un ejemplo del uso de esta sentencia es el siguiente:

```
PARA i<- 1 HASTA N  HACER
        suma <- suma +  vector[i]
FINPARA
```

- *Sentencia REPETIR – HASTA – QUE:* Se trata de una sentencia de control repetitiva que ejecuta siempre al menos una iteración. Entre las palabras reservadas REPETIR y HASTA se colocan las sentencias que se ejecutan en cada iteración. Tras la palabra reservada QUE debe colocarse una expresión lógica. Mientras dicha expresión no tome el

valor CIERTO se seguirán ejecutando las instrucciones del bucle. Un ejemplo de la utilización de esta sentencia es:

```
REPETIR
        s <- s + 1/n
        n <- n-1
HASTA QUE n = 0
```

- *Sentencia CASO – SEA – ENOTROCASO – FINCASO:* La sentencia de selección simple SI –ENTONCES –FINSI nos permite únicamente elegir entre dos alternativas posibles. Sin embargo, en programación se da con frecuencia la necesidad de elegir entre más de dos alternativas de acción. Esto puede lograrse mediante la sentencia CASO. Para ello entre las palabras reservadas CASO y SEA debe aparecer una expresión de tipo ordinal. Tras eso vienen las listas de etiquetas CASO, que deben comenzar con el símbolo "|", seguidas por un conjunto de valores del tipo de la expresión principal separados por comas o un rango entre un valor mínimo y un valor máximo. La lista de etiquetas acaba con dos puntos (":") y continuación a aparecen las sentencias que deben ejecutarse en caso de que la expresión seleccionada tome algunos de los valores anteriores. Los valores empleados en las etiquetas CASO deben ser constantes (no se puede emplear una variable por ejemplo), y no pueden aparecer duplicados. Tras el conjunto de listas etiquetas CASO y las sentencias correspondientes, puede aparecer opcionalmente la palabra reservada ENOTROCASO seguida de las acciones a ejecutar en caso de que la expresión seleccionada tome un valor que no aparezca en ninguna de las listas de etiquetas. Un ejemplo de la utilización de una sentencia CASO es el siguiente:

```
TIPOS
    Tdias = (LUNES,MARTES,MIÉRCOLES,JUEVES,VIERNES,SÁBADO,DOMINGO)
  VARIABLES
    dia:Tdias
  INICIO
      CASO dia SEA
          |LUNES .. VIERNES: Escribir("Día laborable")
          |SÁBADO,DOMINGO: Escribir("Día festivo")
      ENOTROCASO
          Escribir("Día incorrecto")
      FINCASO
```

- *Sentencia Escribir:* Las sentencias de entrada/salida del pseudolenguaje son muy simples y fáciles de usar. En primer lugar la sentencia escribir sirve para sacar por la salida estándar (normalmente monitor), la información que deseemos. La sentencia se compone de la palabra reservada Escribir, y a continuación entre paréntesis ("(" y ")"), una serie de expresiones separadas por comas. Las expresiones pueden ser de cualquier tipo, y también pueden ser cadenas de caracteres encerradas entre comillas simples. Un ejemplo del uso de la sentencia Escribir es el siguiente:

```
VARIABLES
      edad:NATURAL
      nombre:ARRAY [1..N] DE CARÁCTER
INICIO
       ...
      Escribir("Nombre:" , nombre," Edad:",edad)
```

- *Sentencia Leer:* Esta sentencia se usa para asignar a una serie de variables los datos leídos de la entrada estándar (normalmente el teclado). La sentencia se compone de la palabra reservada Leer, y a continuación entre paréntesis la lista de variables que se desean leer. Las variables pueden ser de cualquier tipo. Por ejemplo:

```
VARIABLES
        edad:NATURAL
        nombre:ARRAY [1..N] DE CARÁCTER
INCIO
        Escribir("Introduzca su edad y a continuación su nombre: ")
        Leer(edad,nombre)
```

- *Sentencia Saltar_Linea:* Esta sentencia se usa simplemente para pasar a la siguiente línea cuando estamos escribiendo con la sentencia Escribir. Se debe escribir Saltar_Linea como única palabra reservada de la sentencia.
- *Sentencia RESULTADO:* Esta sentencia sólo puede ser usada dentro de una función y se utiliza para devolver una expresión del mismo tipo que retorna la función. Para ello se escribe primero la palabra reservada RESULTADO (no hace falta declarar una variable con este nombre), a continuación el operador de asignación (<-) y por último la expresión que se quiere devolver. Un ejemplo del uso de esta sentencia es el siguiente:

```
FUNC factorial (n:NATURAL):NATURAL
INICIO
        SI n = 0 ENTONCES
                RESULTADO <- 1
        ENOTROCASO
                RESULTADO <- n * factorial(n-1)
        FINSI
FIN
```

2.7.Comentarios en el pseudolenguaje.

Los comentarios en el pseudolenguaje siguen el mismo formato que en C, es decir, comienzan con una barra y un asterisco (/*) y terminan con un asterisco y una barra (*/). Los comentarios

pueden tener varias líneas siempre que comiencen y terminen como se ha indicado antes. Por ejemplo:

```
/* Esto es un  comentario
   válido en el pseudolenguaje
   con varias líneas */
```

Sin embargo, no están permitidos los comentarios anidados, ya que empeoran la legibilidad del código y no aportan ningún beneficio. Así, por ejemplo, el siguiente comentario produciría un error sintáctico:

```
/* Esto es un  comentario
   del pseudolenguaje
/* anidado */  no valido*/
```

2.8. Procedimientos y funciones en el pseudolenguaje.

Como ya se ha explicado, la definición de subalgoritmos en el pseudolenguaje debe empezar con la palabra reservada FUNC en el caso de las funciones y con PROC en el caso de los procedimientos seguidas de un identificador que indique el nombre del subalgoritmo. A continuación deben aparecer entre paréntesis los parámetros formales del subalgoritmo. La sintaxis de la declaración de parámetros formales es prácticamente la misma que de la declaración de variables, es decir, los nombres de las variables separados por comas y tras esto dos puntos y el tipo. Sin embargo, en el caso de los parámetros formales, se deben separar por punto y coma la declaración de parámetros de distinto tipo. Además, en el caso de que los parámetros se pasen por referencia, se debe añadir antes del identificador del parámetro el símbolo ^.

Cuando se quiera invocar un procedimiento o una función, se deben escribir el nombre del mismo seguido de tantas expresiones como parámetros formales tuviera el subalgoritmo. Estas expresiones deben estar separadas por comas y ser del mismo tipo que el parámetro formal

correspondiente. La invocación de un procedimiento constituye por si sola una sentencia del pseudolenguaje. En cambio, la invocación de una función genera una expresión del tipo que devuelve dicha función. El siguiente ejemplo lo ilustra:

```
PROC Escribir_Menu()
INICIO
  ...
FIN

FUNC Realizar_Op(Op1,Op2:ENTERO):REAL
INICIO
  ...
FIN

INICIO
 Escribir_Menu()
 total <- Realizar_Op(1,5)
```

En cuanto al ámbito de las símbolos (constantes, tipos, variables) definidos en los subalgoritmos y en el algoritmo principal podemos decir que:

- El código del algoritmo principal no puede acceder a ningún símbolo definido en los subalgoritmos, solo a los símbolos definidos en el algoritmo principal.
- Los subalgoritmos pueden acceder tanto a sus símbolos propios (no a los definidos en otro subalgoritmo) como a los del algoritmo principal
- En caso de que el nombre de un símbolo definido en un subalgoritmo sea el mismo que el de un símbolo del algoritmo principal, cuando aparezca este símbolo en el código del subalgoritmo se referirá al definido dentro del subalgoritmo.

La siguiente tabla enumera los nombres y las expresiones regulares de todos los tokens empleados en el análisis lexicográfico del pseudolenguaje:

Nombre del token	**Expresión regular**
<algoritmo>	“ALGORITMO”
<proc>	“PROC”
<func>	“FUNC”
<resultado>	“RESULTADO”
<punto_y_coma>	“;”
<coma>	“,”
<dos_puntos”	“:”
<punto>	“.”
<tipos>	“TIPOS”
<variables>	“VARIABLES”
<constantes>	“CONSTANTES”
<natural>	“NATURAL”
<entero>	“ENTERO”
<real>	“REAL”
<puntero>	“PUNTERO”
<fichero>	“FICHERO”
<a>	“A”
<nuevo>	“NUEVO”
<eliminar>	“ELMININAR”
<nulo>	“NULO”
<logico>	“LOGICO”
<caracter>	“CARACTER”
<array>	“ARRAY”
<de>	“DE”
<registro>	“REGISTRO”
<finregistro>	“FINREGISTRO”

<inicio>	“INICIO”
<fin>	“FIN”
<falso>	“FALSO”
<chr>	“CHR”
<cap>	“CAP”
<ord>	“ORD”
<pred>	“PRED”
<suc>	“SUC”
<abrir>	“ABRIR”
<anadir>	“ANADIR”
<crear>	“CREAR”
<cerrar>	“CERRAR”
<leerbin>	“LEERBIN”
<escribirbin>	“ESCRIBIRBIN”
<fin_fichero>	“EOF”
<LEER>	“LEER”
<ESCRIBIR>	“ESCRIBIR”
<cierto>	“CIERTO”
<op_logico_or>	“\\/”
<op_logico_and>	“/\\”
<op_logico_not>	“\u00ac”
<op_puntero>	“^”
<menos>	“-“
<mas>	“+”
<por>	“*”
<entre>	“/”
<modulo>	“MOD”
<op_asignación>	“<-“
<rango>	“..”
<par_izq>	“(“

<par_der>	“)”
<cor_izq>	“[“
<cor_der>	“]”
<barra>	“|”
<eq>	“=”
<neq>	“#”
<gt>	“>”
<lt>	“<”
<le>	“<=”
<ge>	“>=”
<fincad>	“FINCAD”
<si>	“SI”
<entonces>	“ENTONCES”
<finsi>	“FINSI”
<enotrocaso>	“ENOTROCASO”
<mientras>	“MIENTRAS”
<hacer>	“HACER”
<finmientras>	“FINMIENTRAS”
<repetir>	“REPETIR”
<hasta>	“HASTA”
<que>	“QUE”
<caso>	“CASO”
<sea>	“SEA”
<fincaso>	“FINCASO”
<para>	“PARA”
<paso>	“PASO”
<finpara>	“FINPARA”
<leer>	“Leer”
<escribir>	“Escribir”
<saltar_linea>	“Saltar_Linea”

<identificador>	[“a”-“z”,”A”-“Z”,”_”] ([“a”-“z”,”A”-“Z”,”_”,”0”-“9”])*
<constante>	([“0”-“9”])+
<constante_real>	<constante> “.” <constante>
<constante_caracter>	“\”” ~[“\””] “\””
<cadena>	“\”” (~[“\””])* “\””
<posición>	“POSICION”
<borrardir>	“BORRARDIR”
<creardir>	“CREARDIR”
<existe>	“EXISTE”
<borrar>	“BORRAR”
<renombrar>	“RENOMBRAR”
<longitud>	“LONGITUD”
<buscar>	“BUSCAR”
<tamano>	“Tamano”

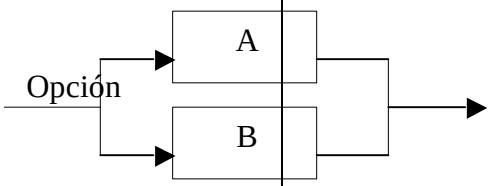
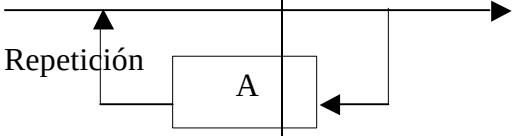
Como puede observarse, casi todas las palabras reservadas del pseudolenguaje aparecen en mayúsculas, y dado que no se ha utilizado la opción IGNORE_CASE, se deben emplear también en mayúsculas cuando se escriba el código, sino se producirá un error léxico.

APÉNDICE I: GRAMÁTICA DEL PSEUDOLENGUAJE

1. Diagramas de Conway

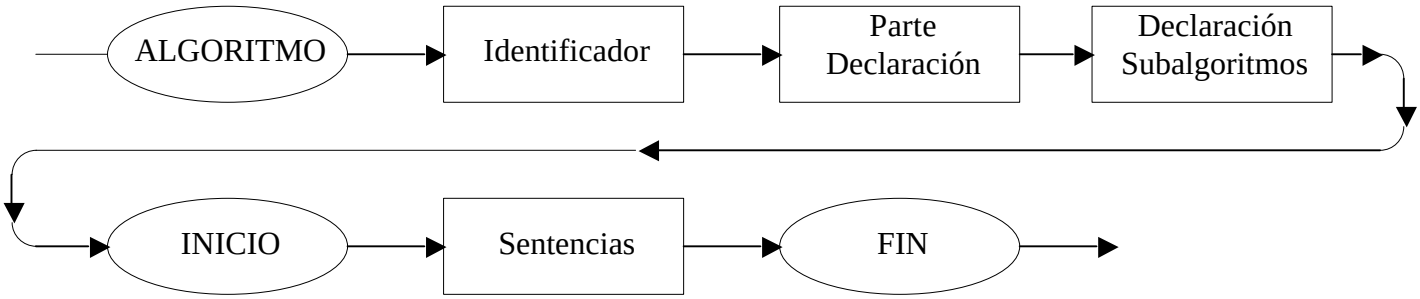
Para la especificación de la gramática del pseudolenguaje usado en este proyecto, se utilizarán diagramas de Conway. Un diagrama de Conway es un grafo dirigido donde los elementos no terminales de una gramática aparecen como rectángulos y los terminales como círculos.

Las operaciones básicas que pueden expresarse mediante los diagramas de Conway son:

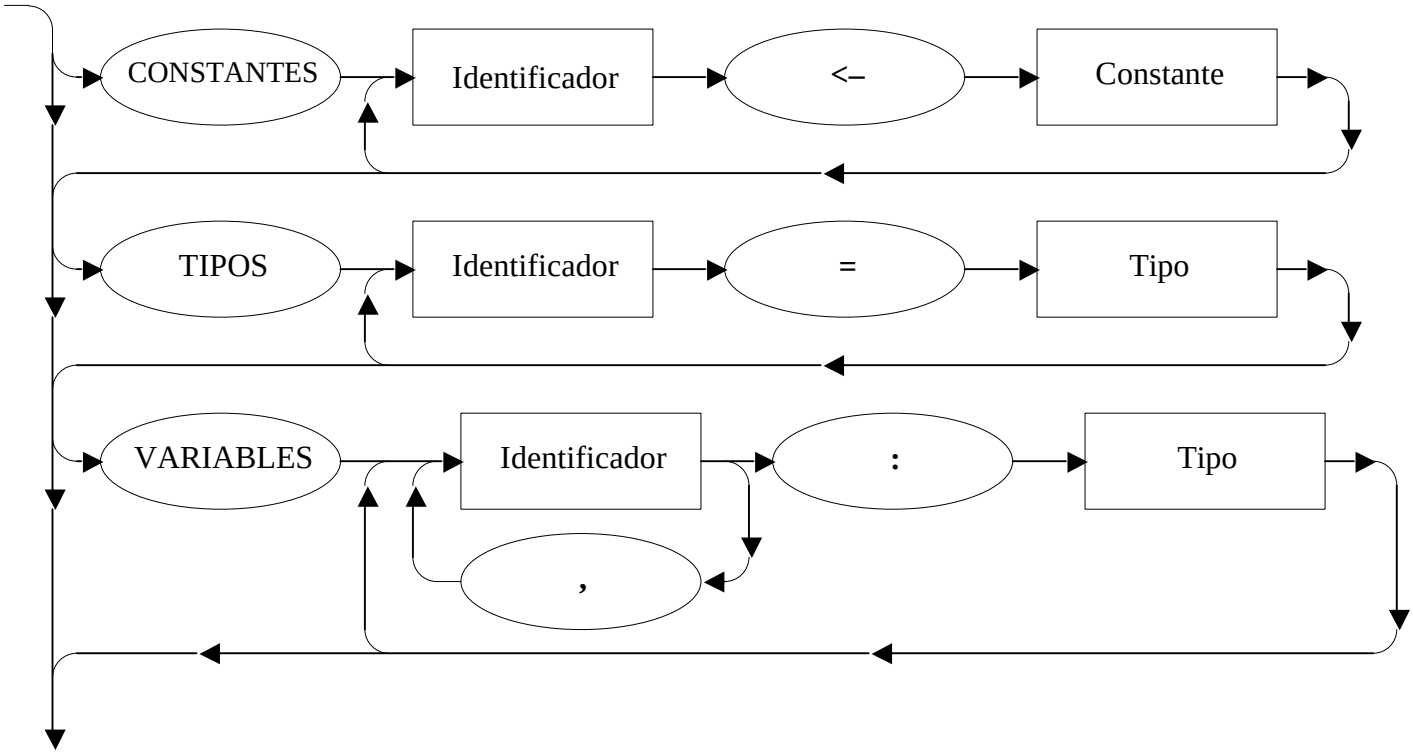
Operación	**Diagrama de Conway**
Yuxtaposición	
Opción	
Repetición	

2. Definición de los elementos del pseudolenguaje mediante diagramas de Conway.

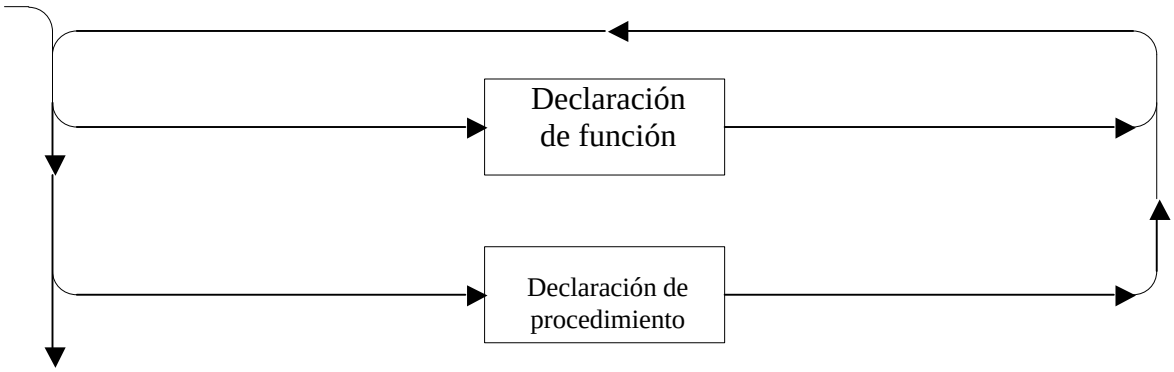
Algoritmo:



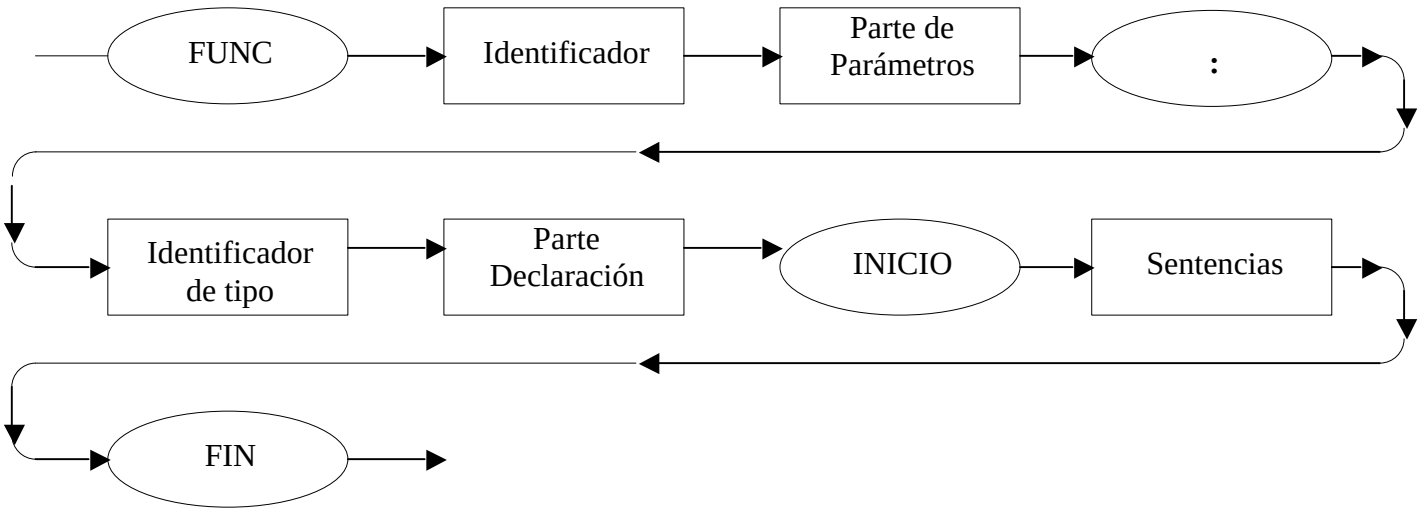
Parte Declaración:



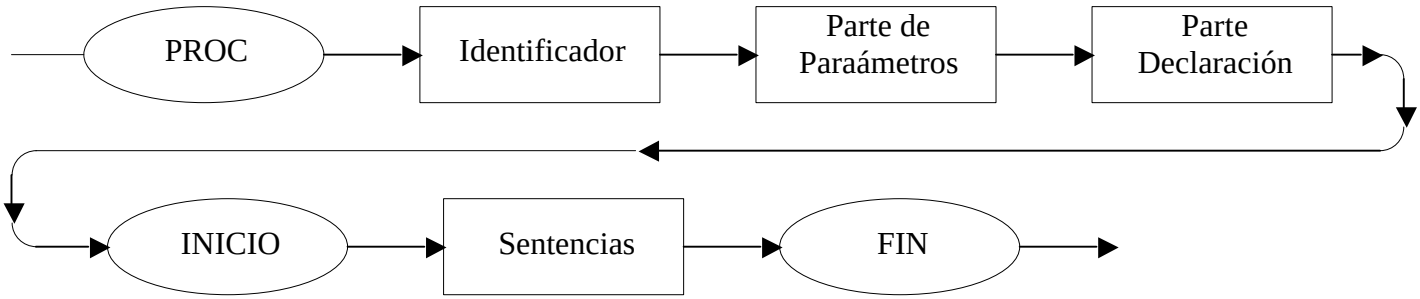
Declaración Subalgoritmos:



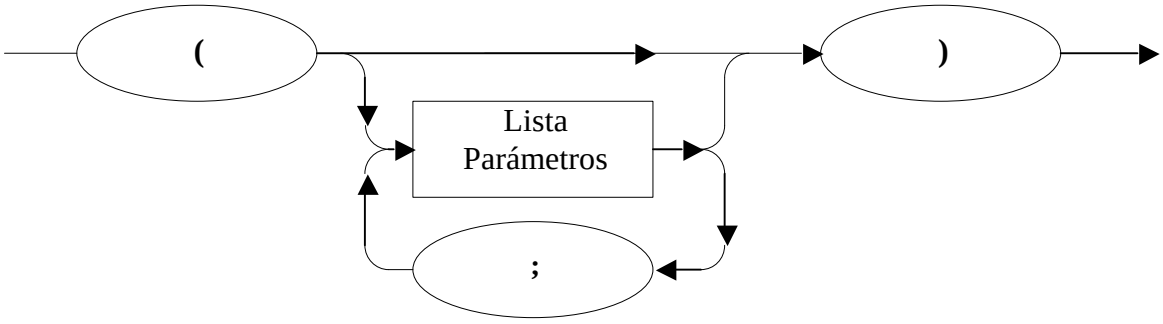
Declarción de función:



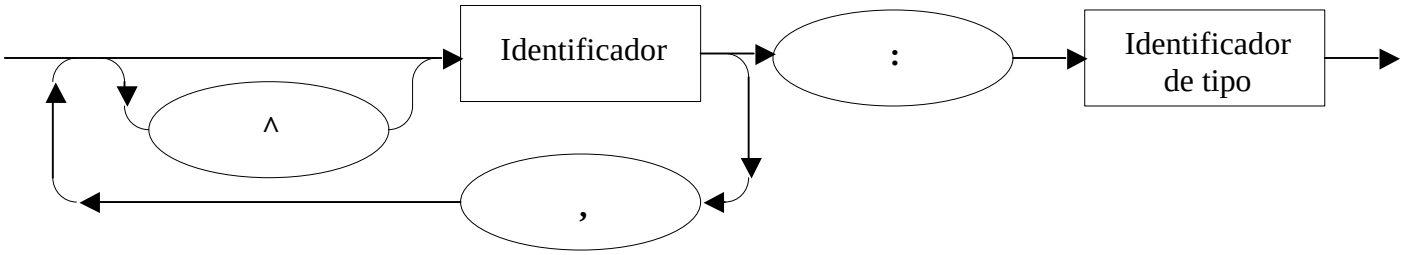
Declaración de procedimiento:



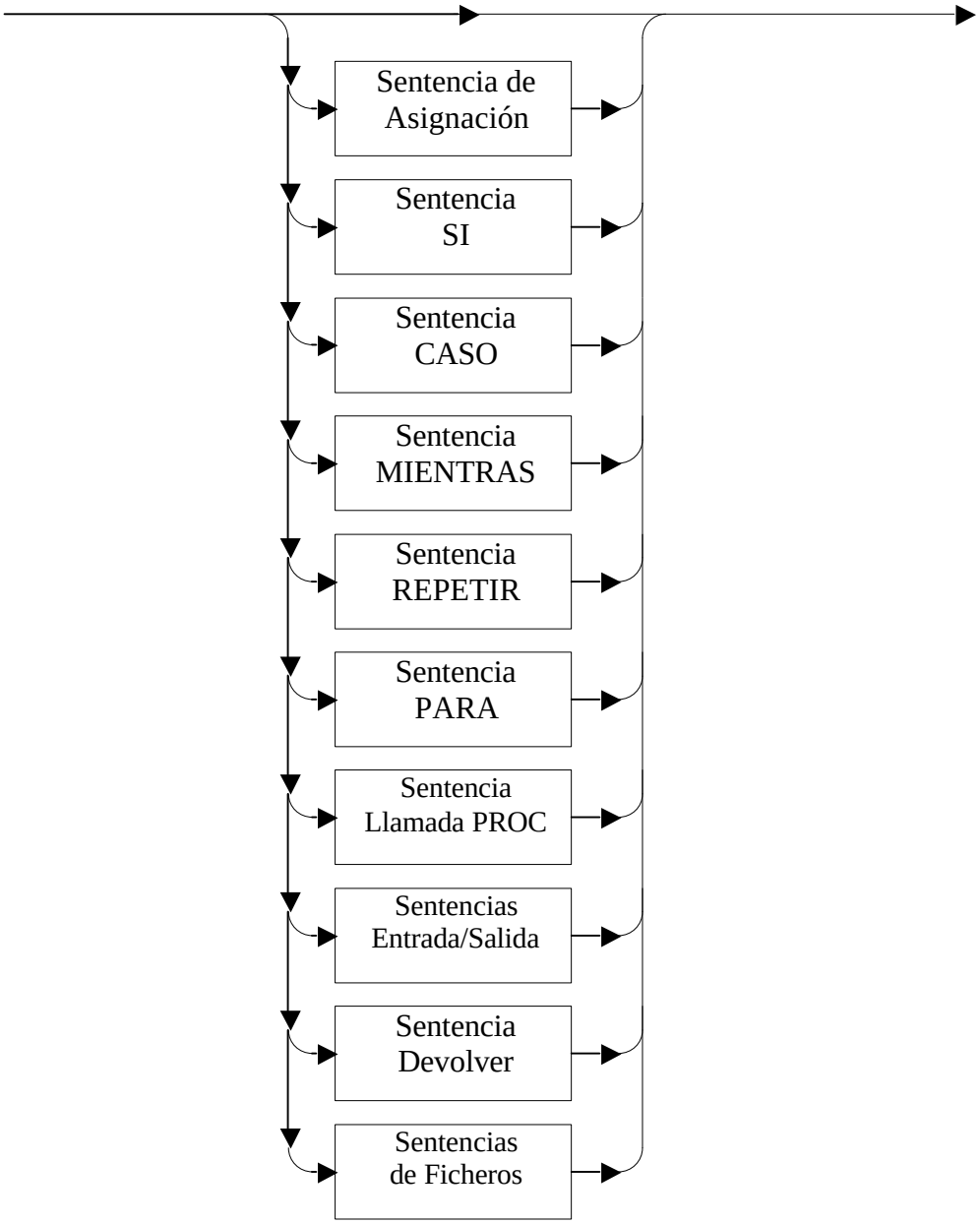
Parte de parámetros:



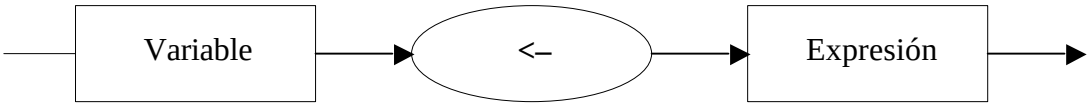
Lista Parámetros:



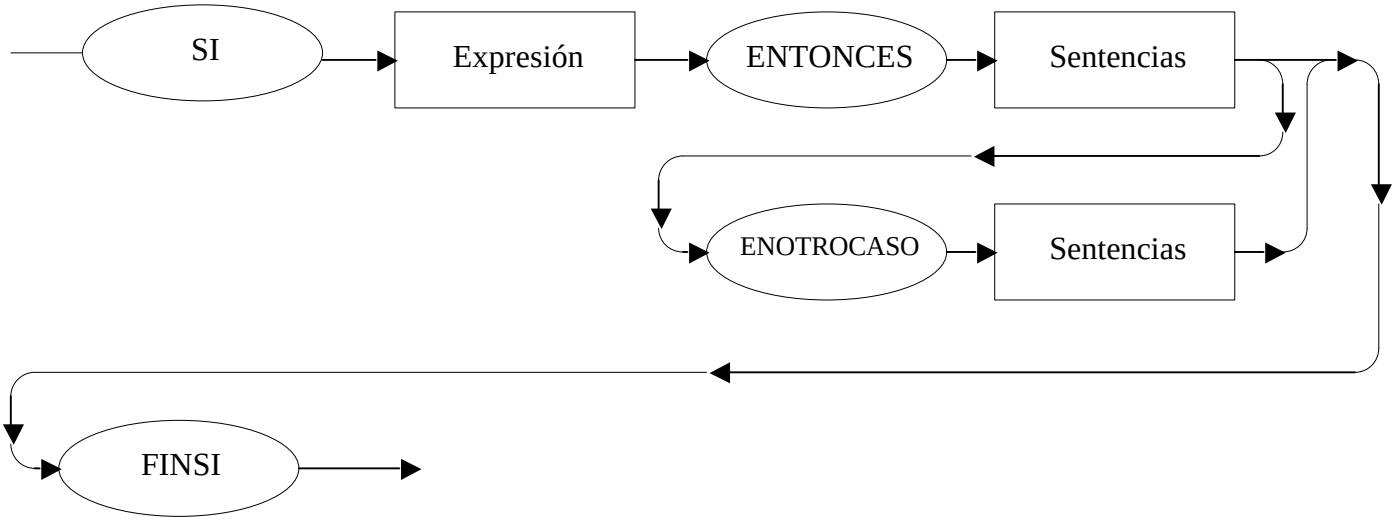
Sentencias:



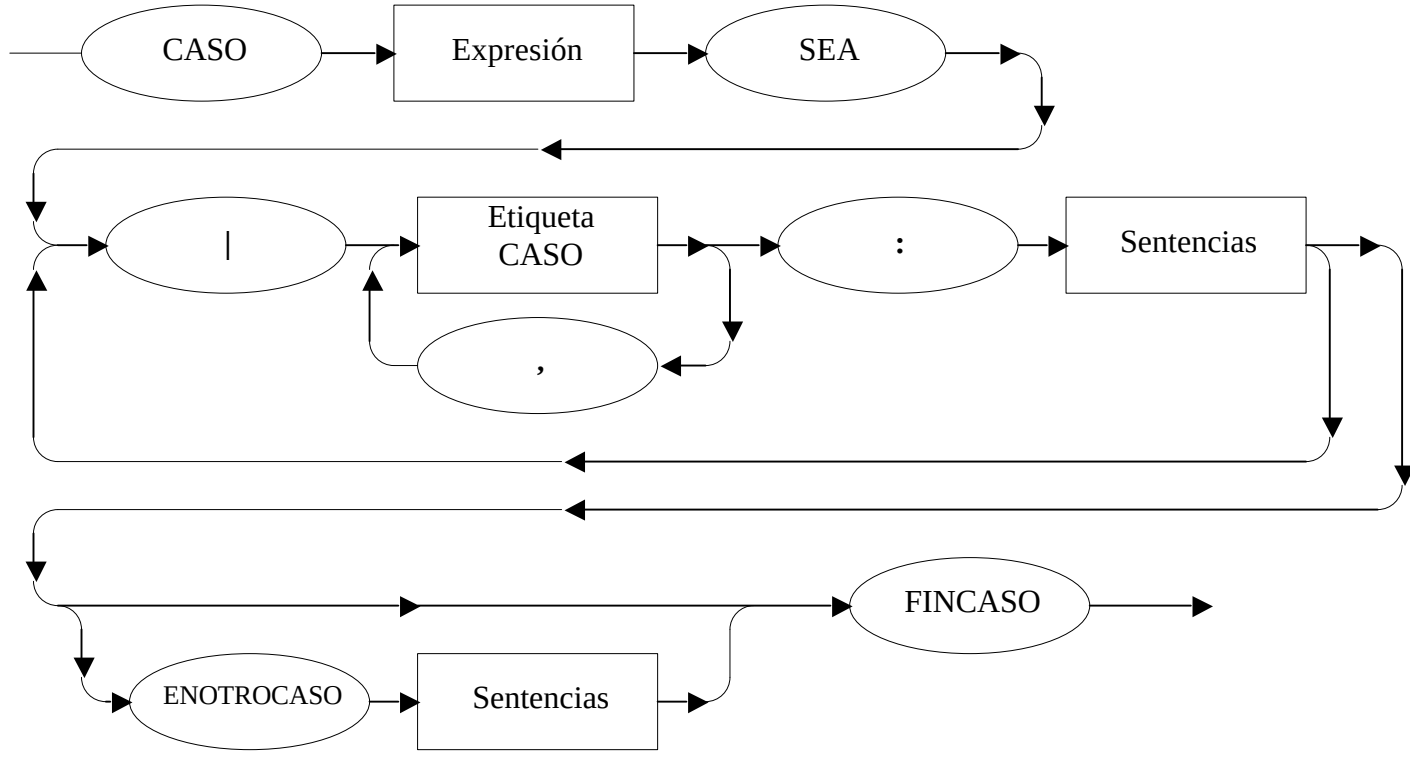
Sentencia de Asignación:



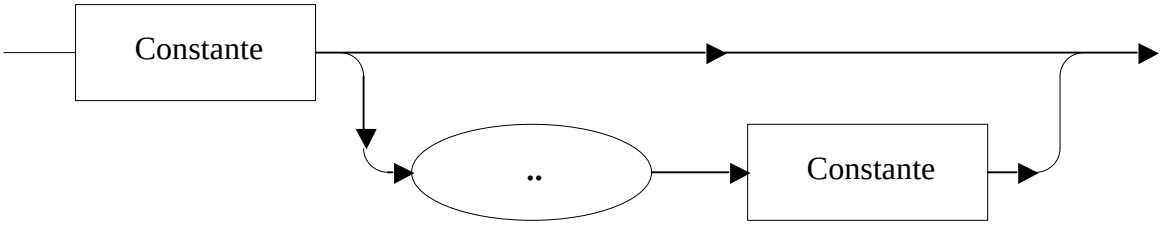
Sentencia SI:



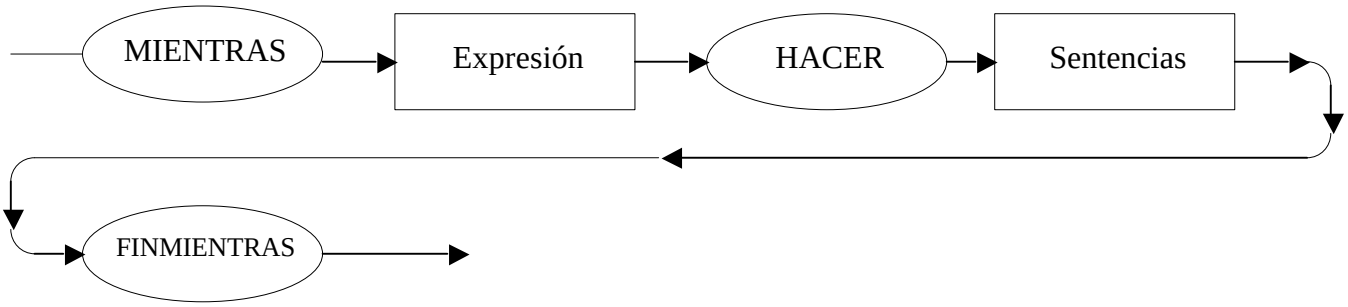
Sentencia CASO:



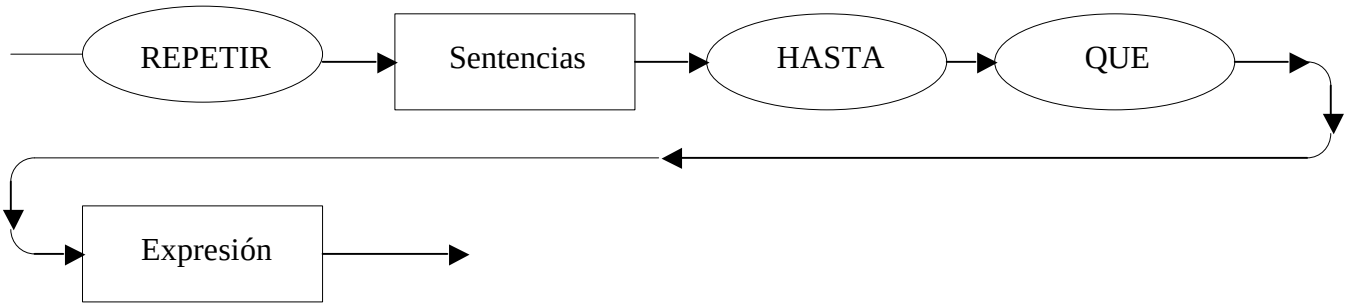
Etiqueta CASO:



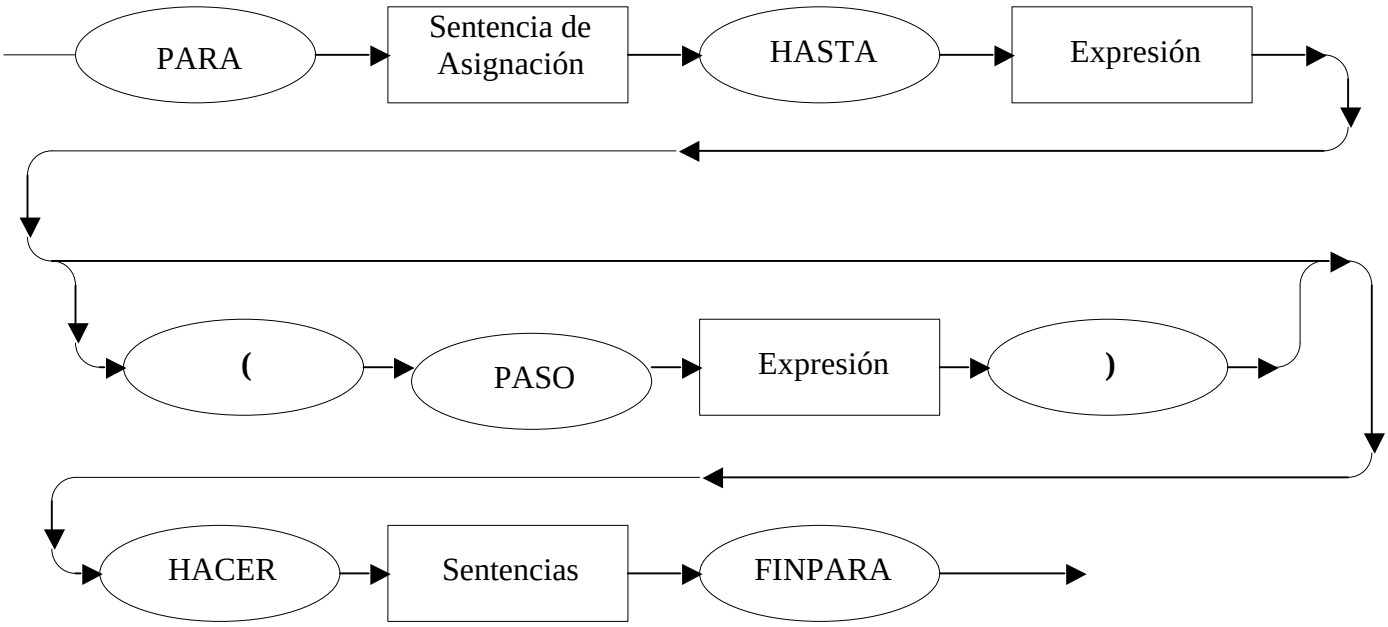
Sentencia MIENTRAS:



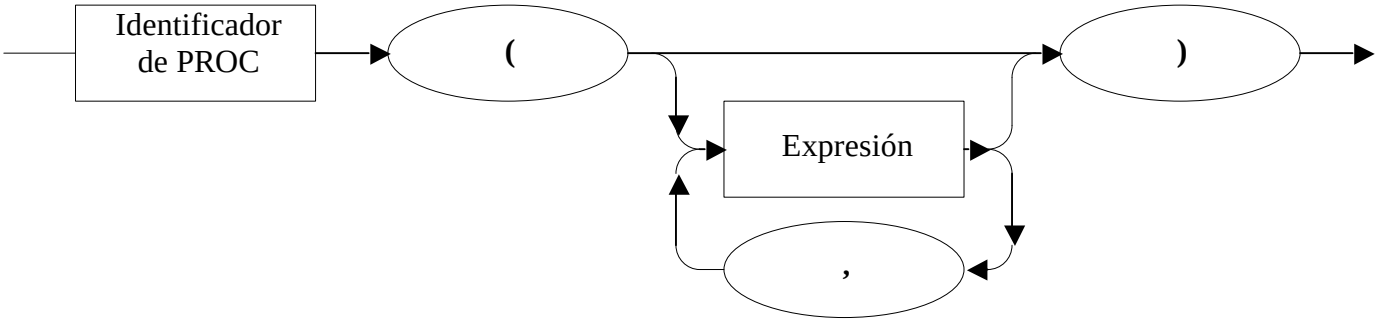
Sentencia REPETIR:



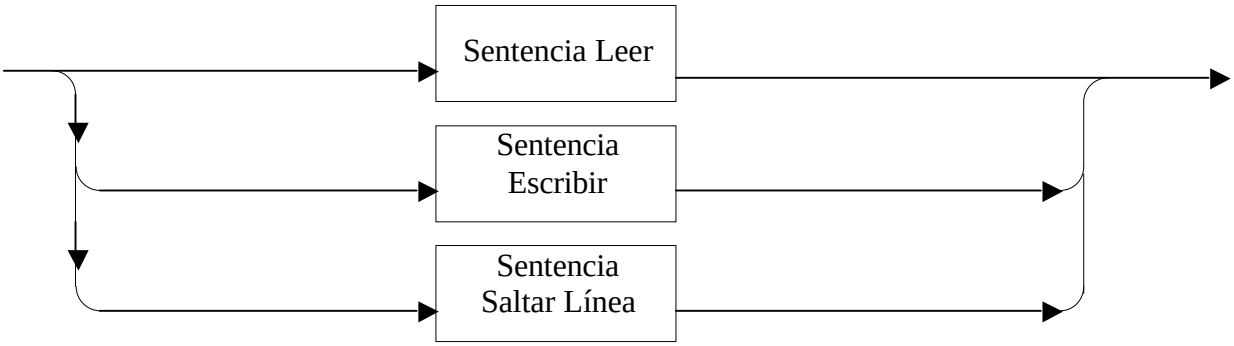
Sentencia PARA:



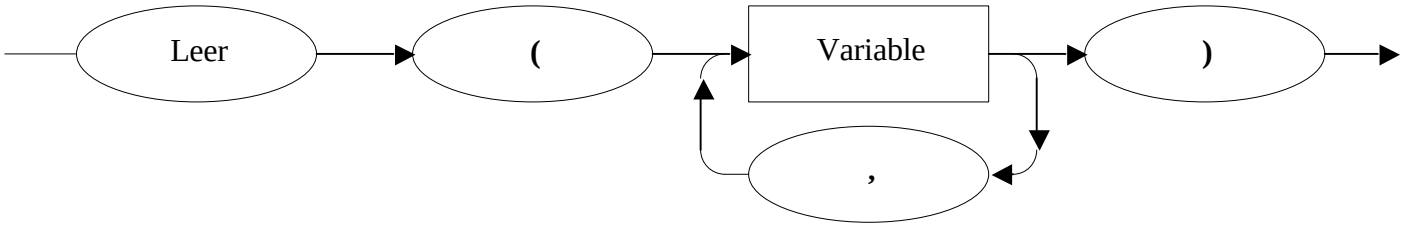
Sentencia Llamada PROC:



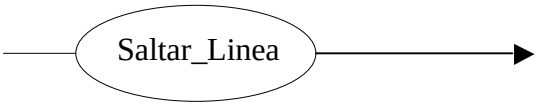
Sentencias Entrada/Salida:



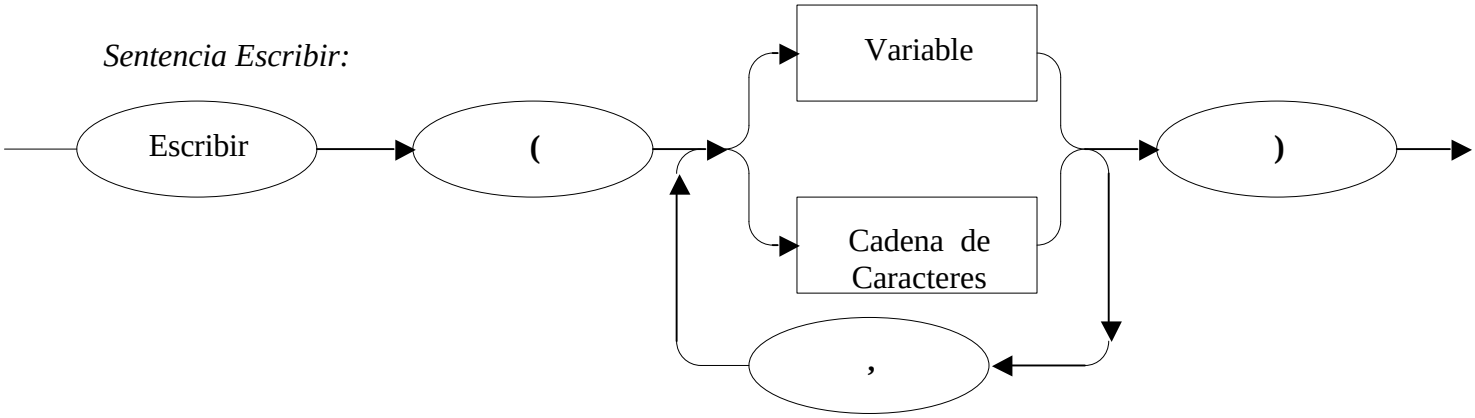
Sentencia Leer:



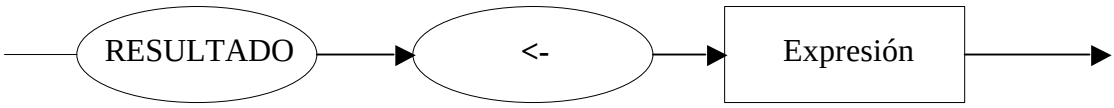
Sentencia Saltar Línea:



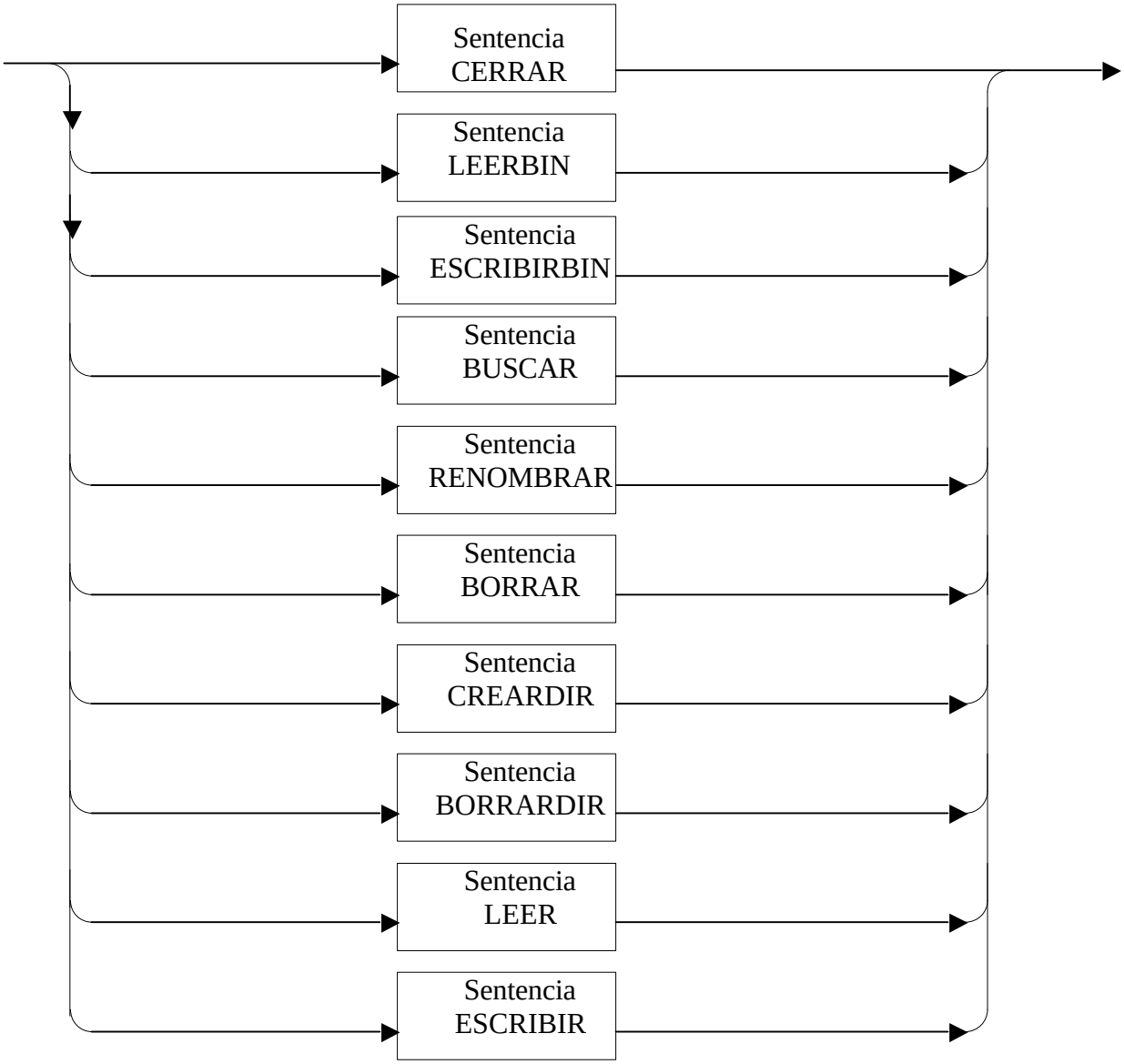
Sentencia Escribir:



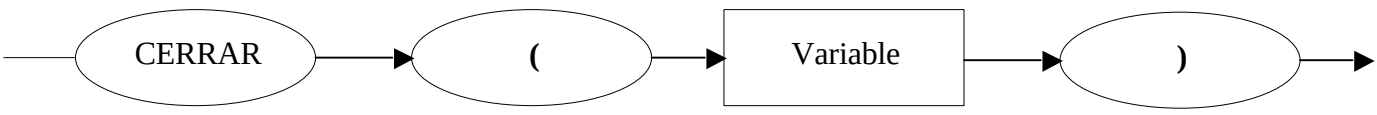
Sentencia Devolver:



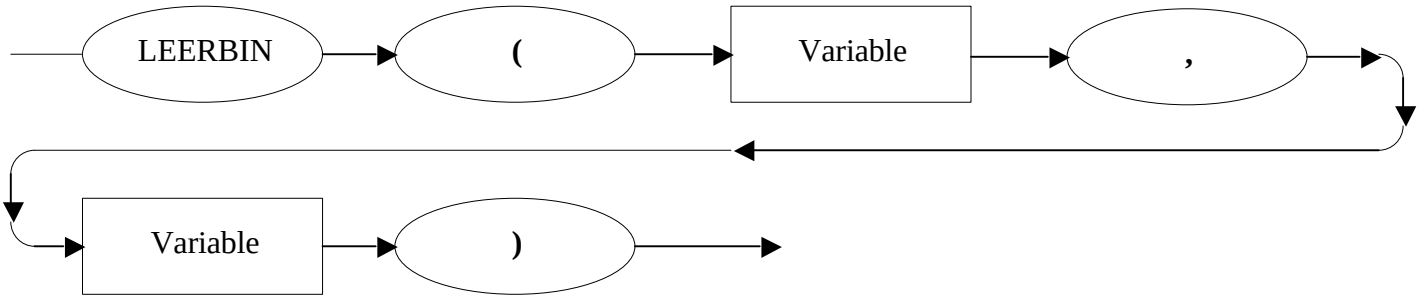
Sentencias Ficheros:



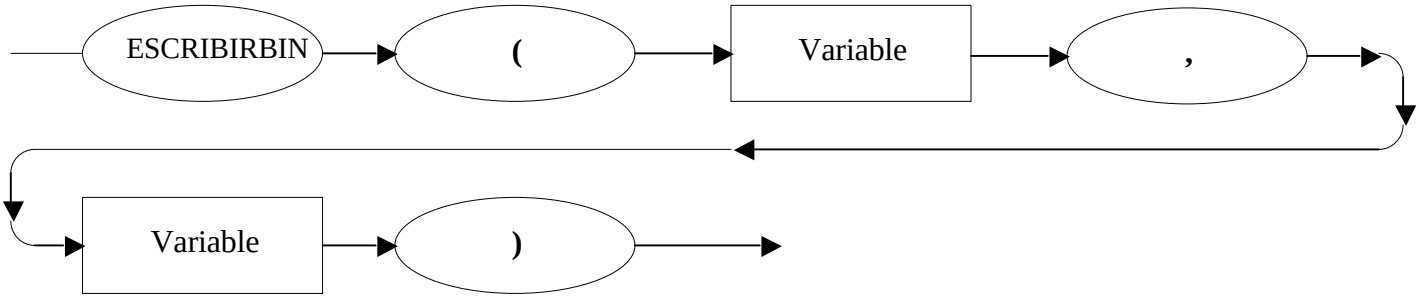
Sentencia CERRAR:



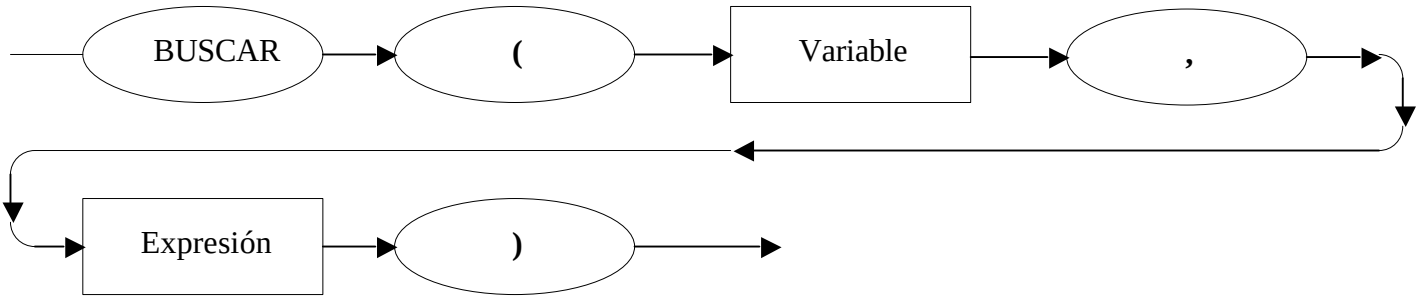
Sentencia LEERBIN



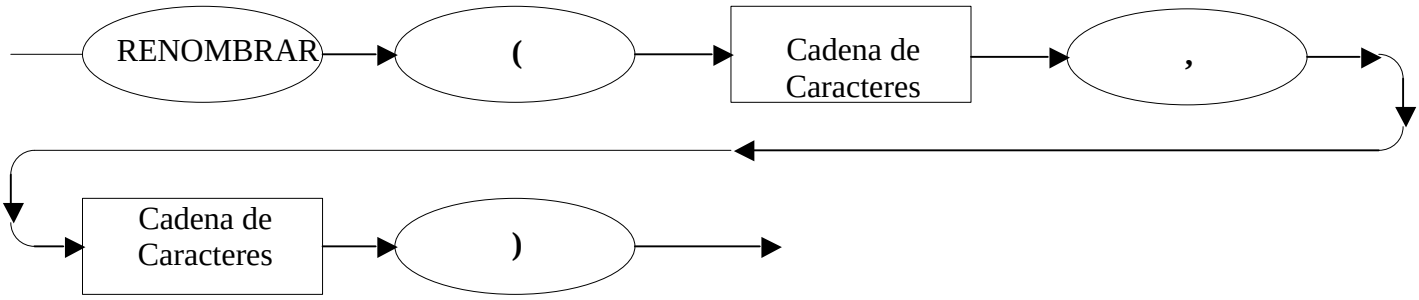
Sentencia ESCRIBIRBIN:



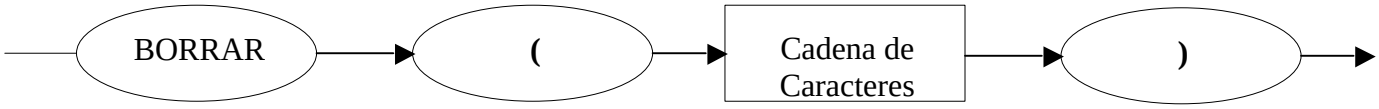
Sentencia BUSCAR:



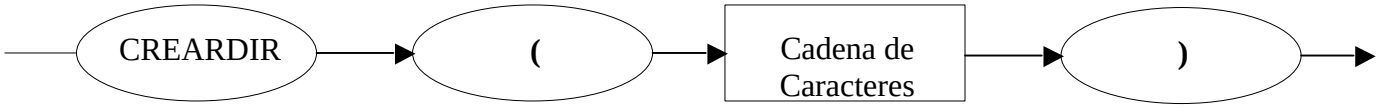
Sentencia RENOMBRAR:



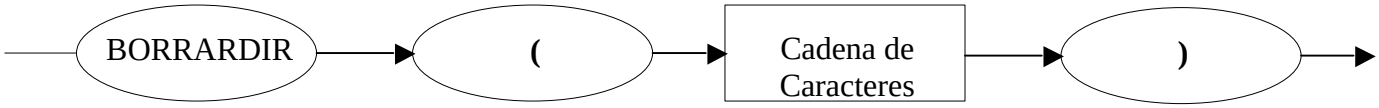
Sentencia BORRAR:



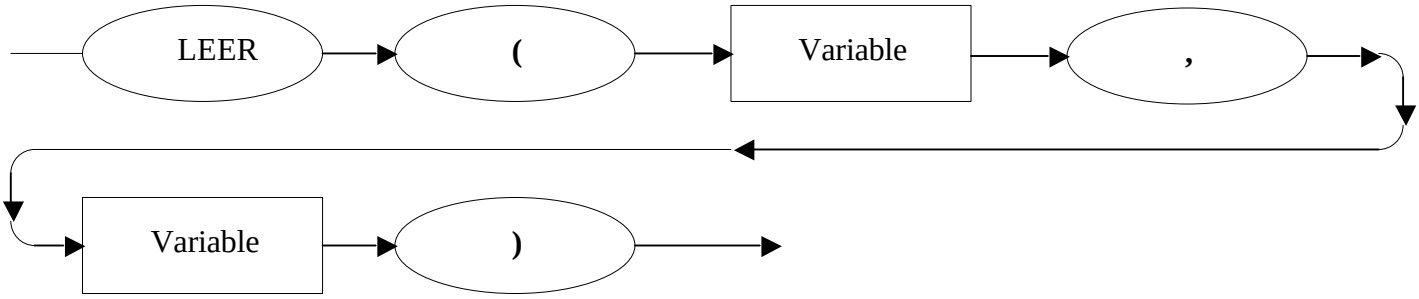
Sentencia CREARDIR:



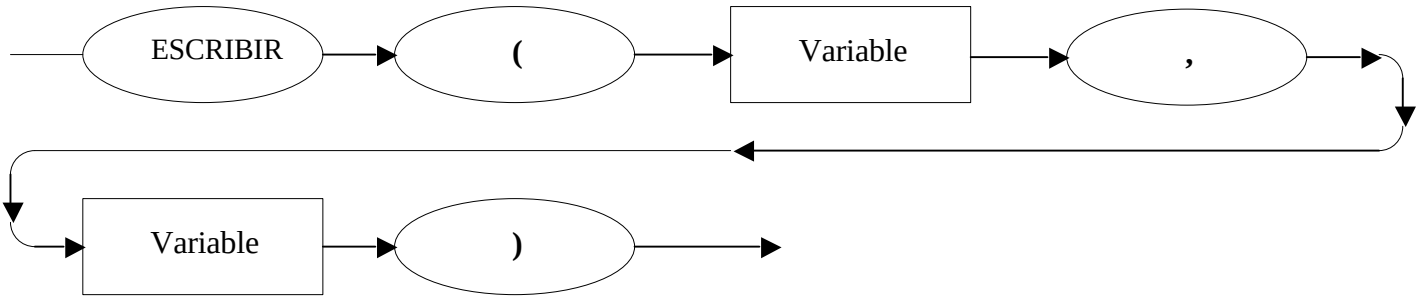
Sentencia BORRARDIR:



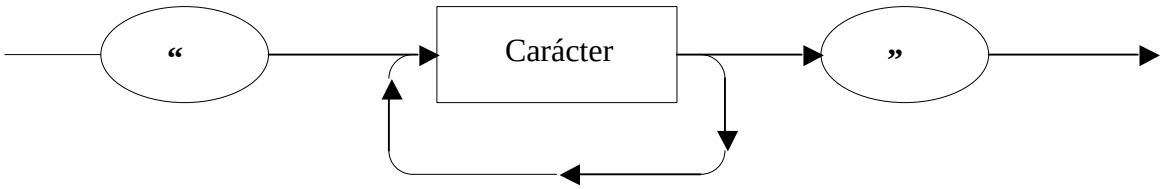
Sentencia LEER:



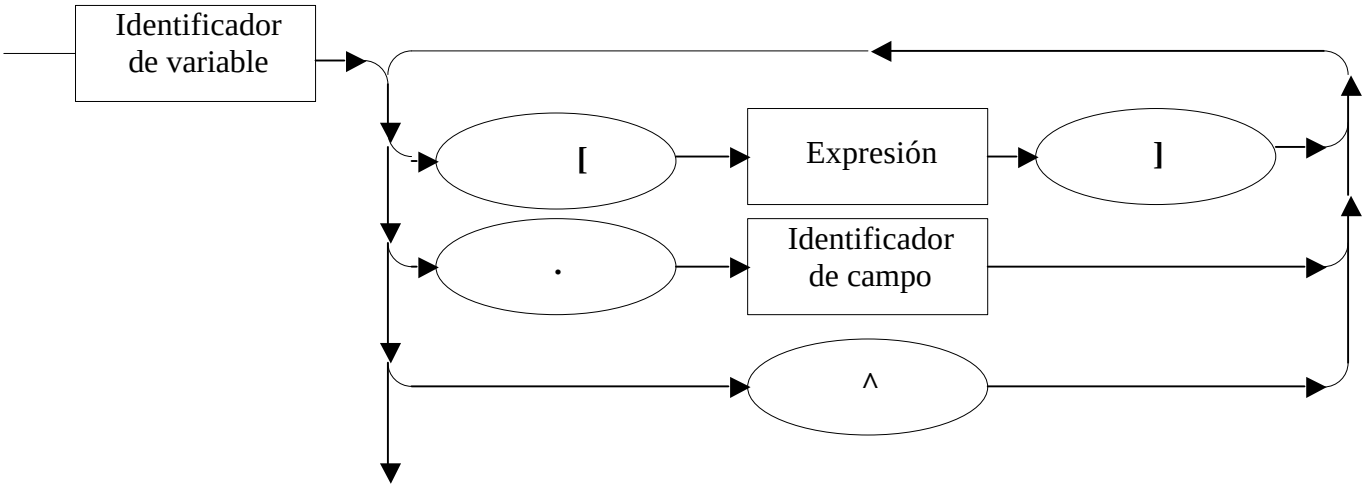
Sentencia ESCRIBIR:



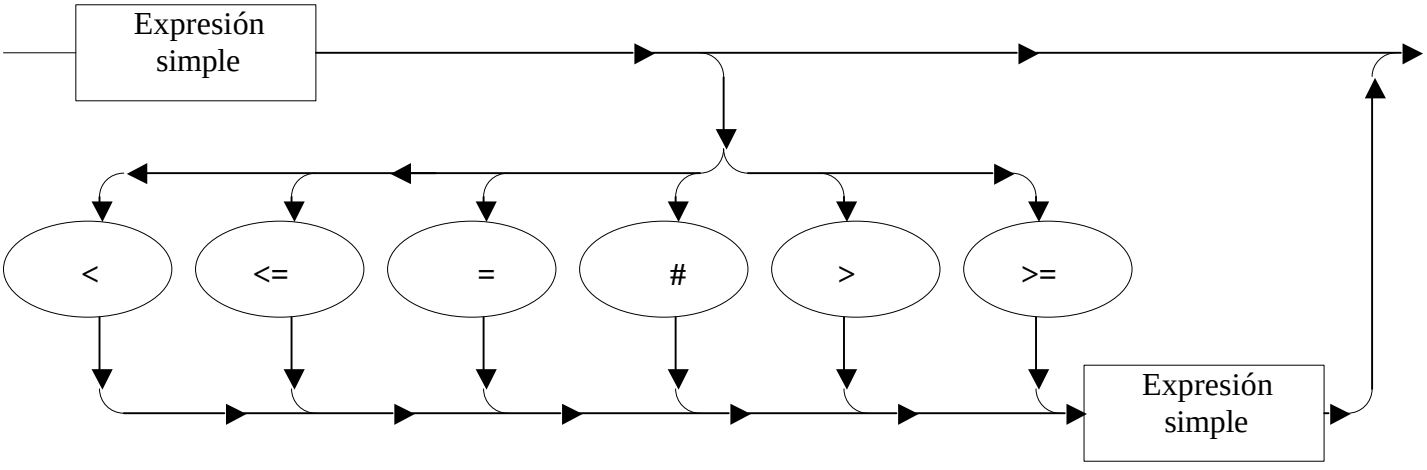
Cadena de Caracteres:



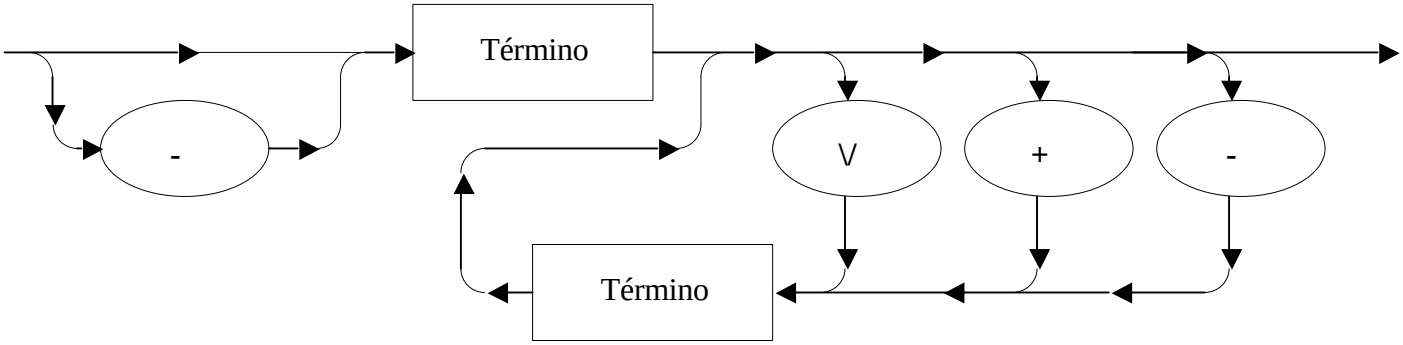
Variable:



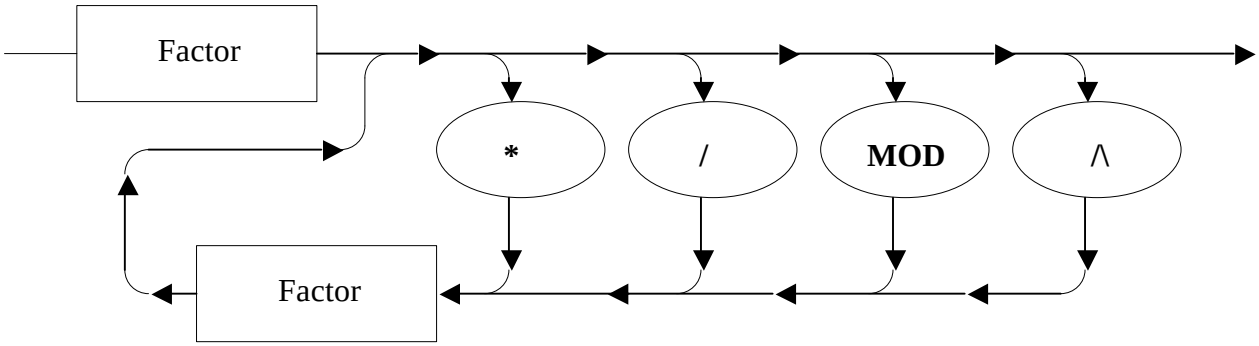
Expresión:



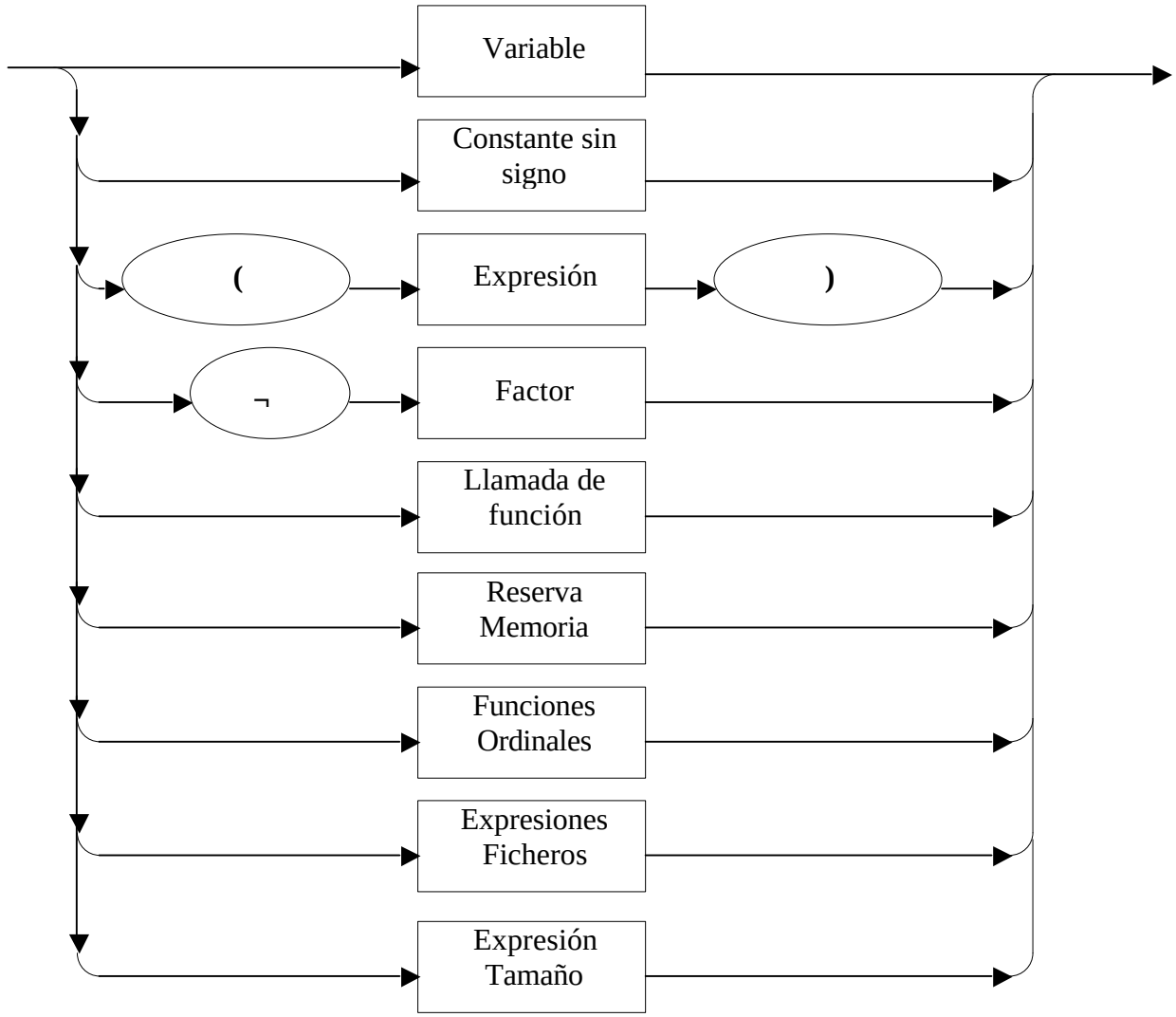
Expresión simple:



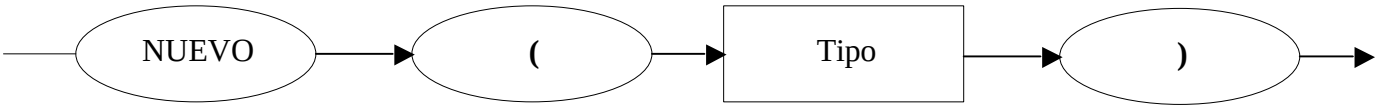
Término:



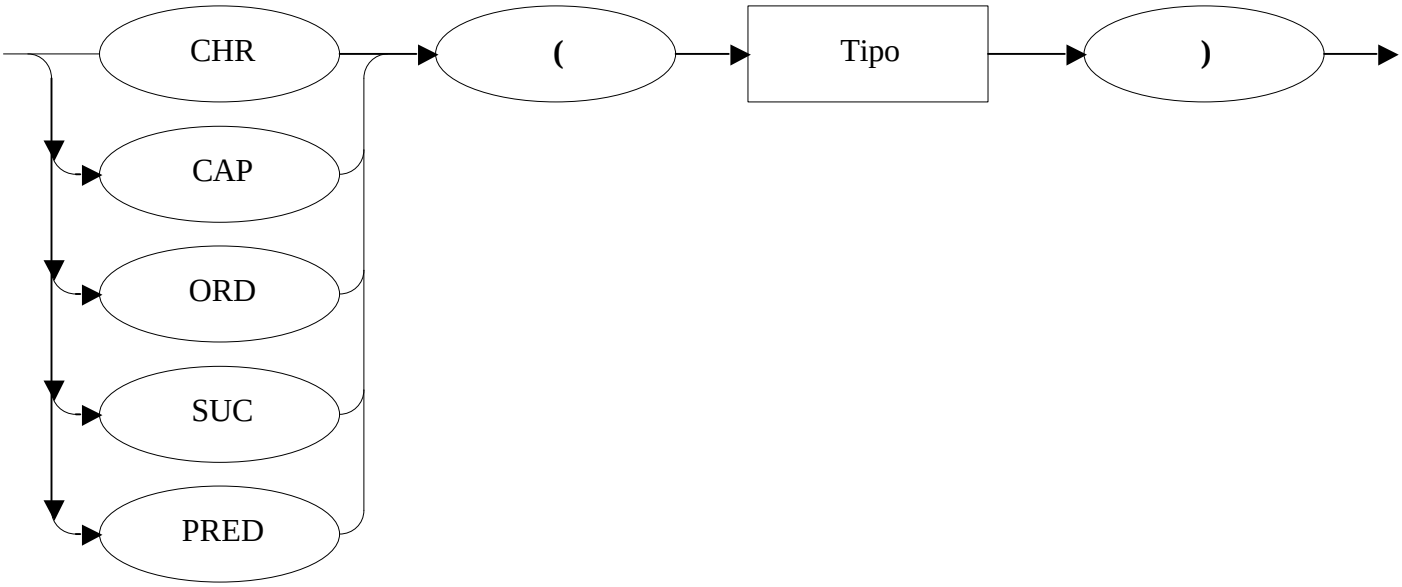
Factor:



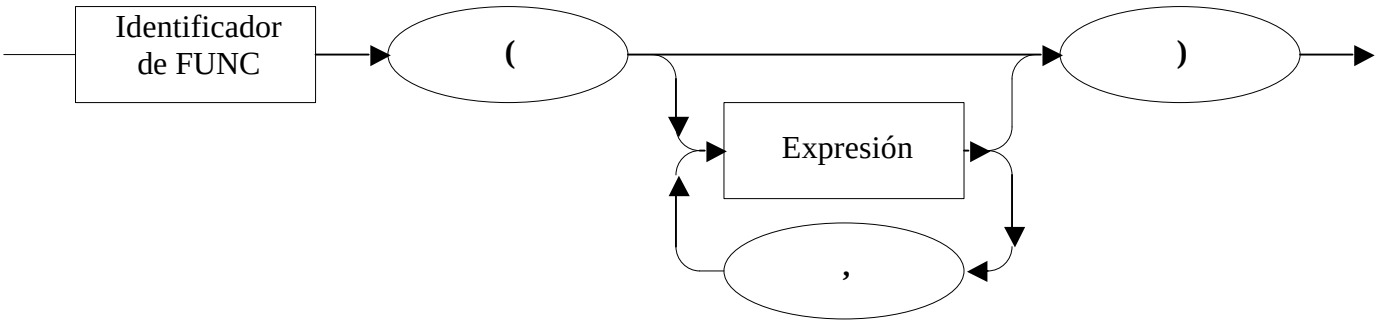
Reserva Memoria:



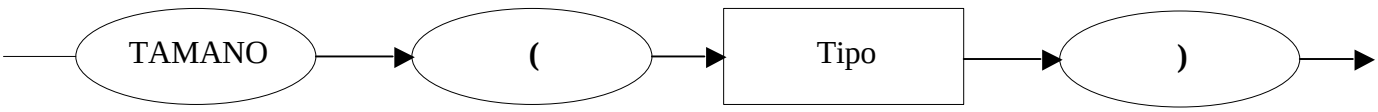
Funciones Ordinales



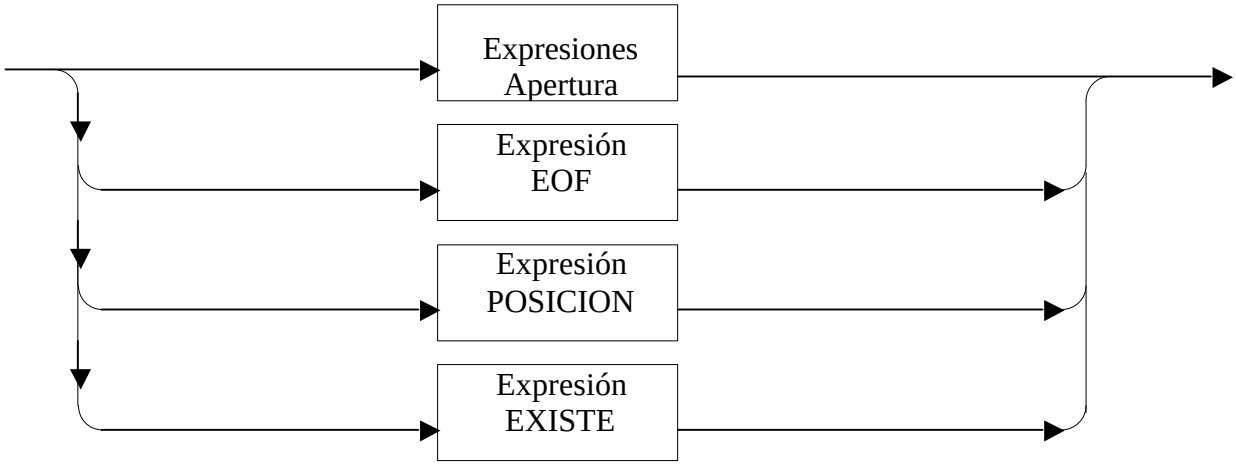
Llamada de función:



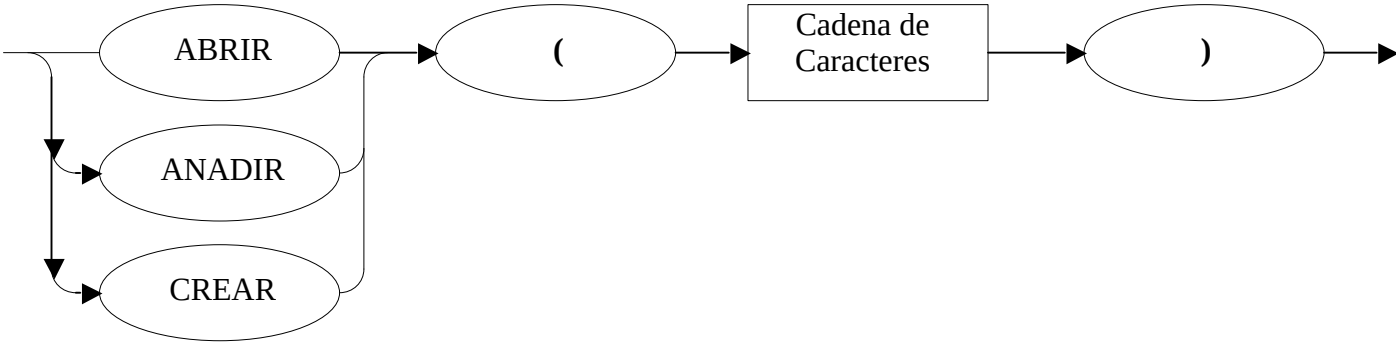
Expresión Tamaño:



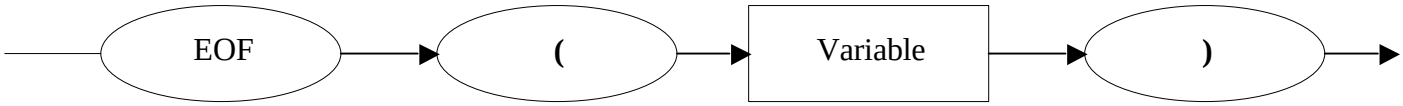
Expresiones Ficheros:



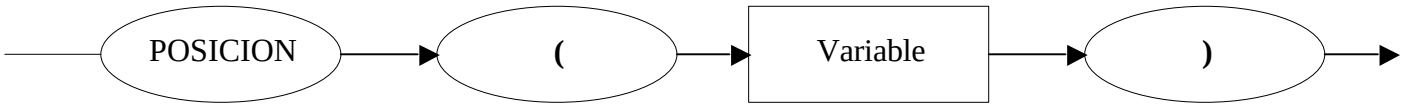
Expresiones Apertura:



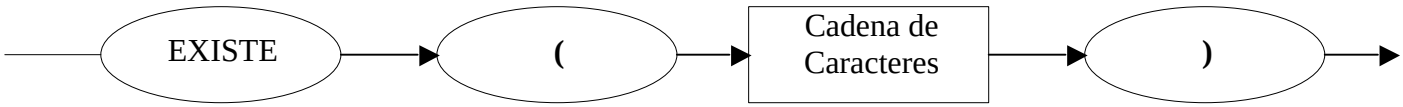
Expresión EOF:



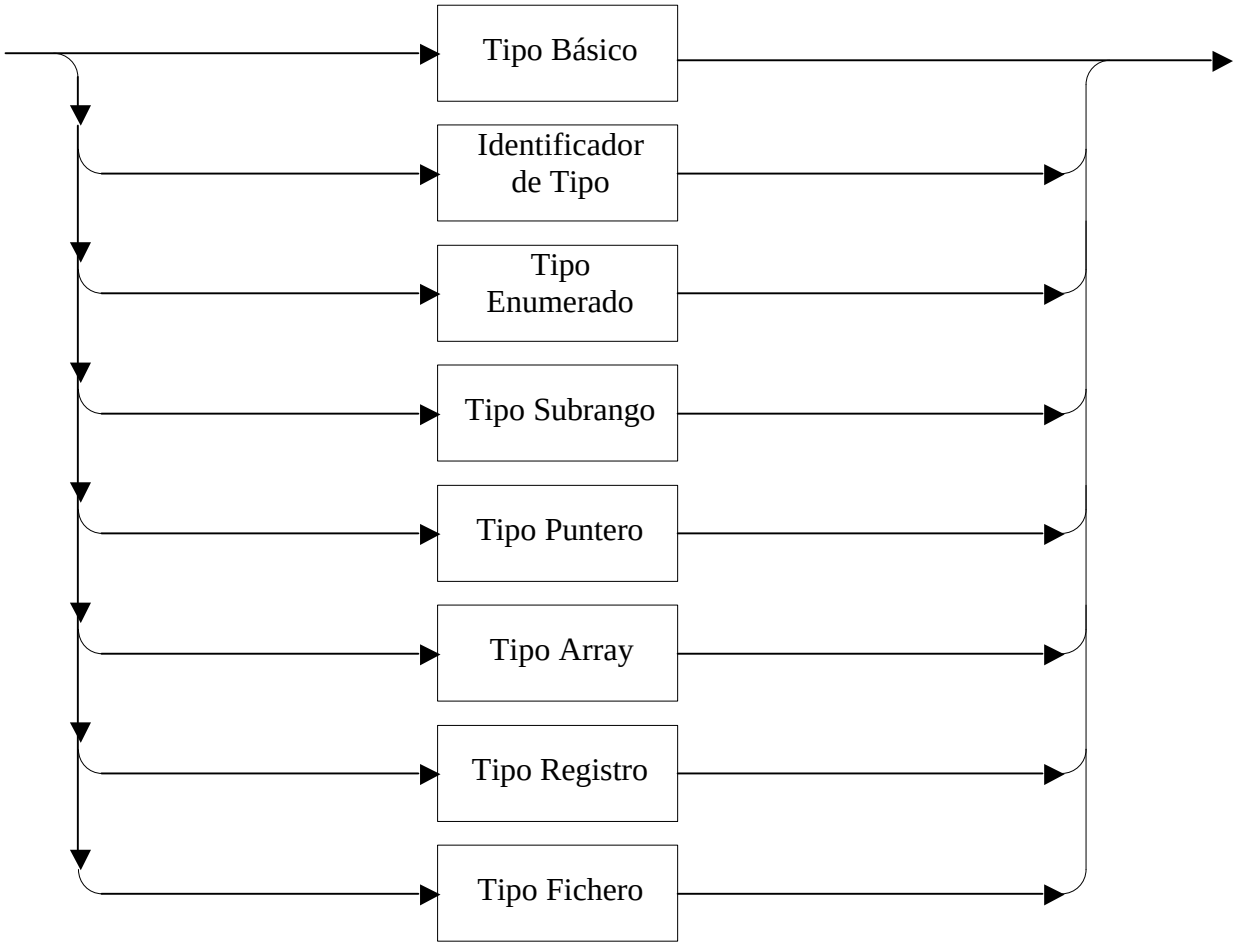
Expresión POSICION:



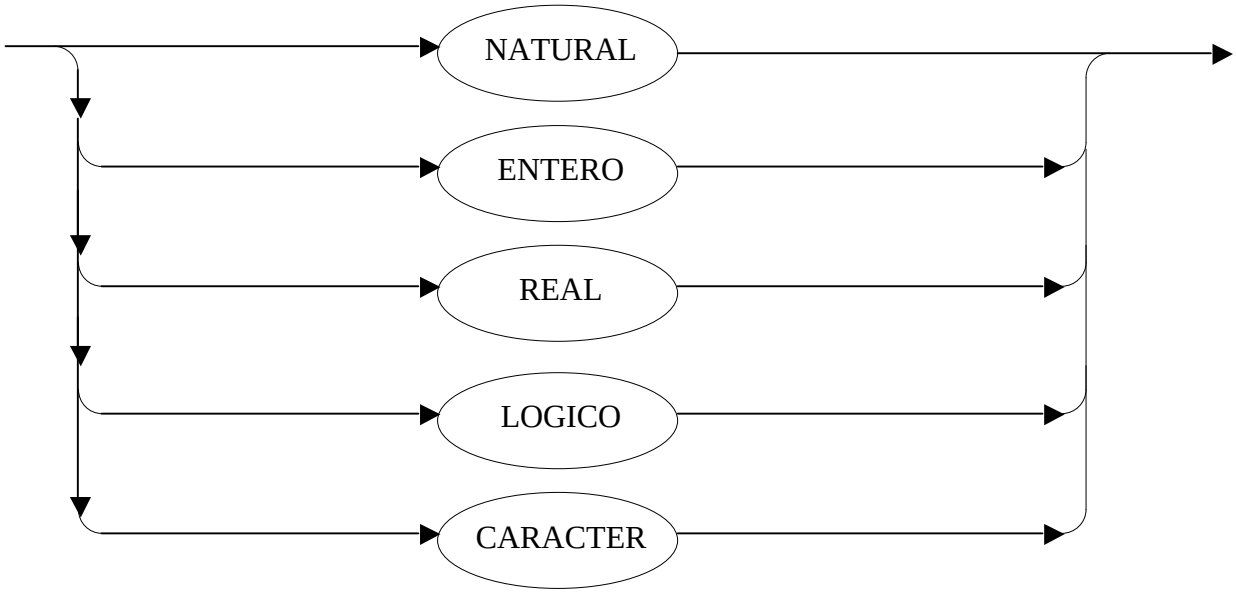
Expresión EXISTE:



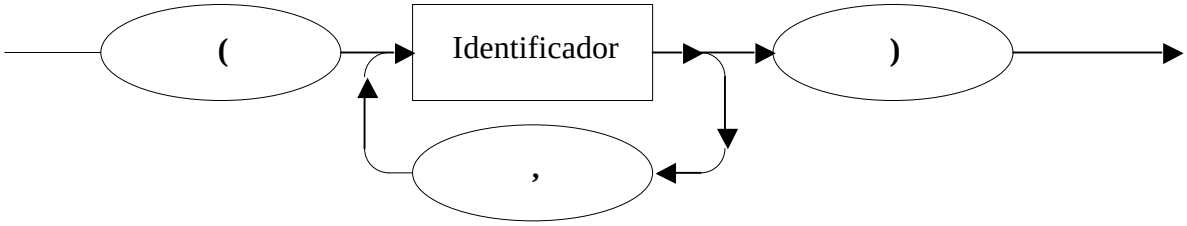
Tipo:


Tipo Básico
Identificador de Tipo
Tipo Enumerado
Tipo Subrango
Tipo Puntero
Tipo Array
Tipo Registro
Tipo Fichero

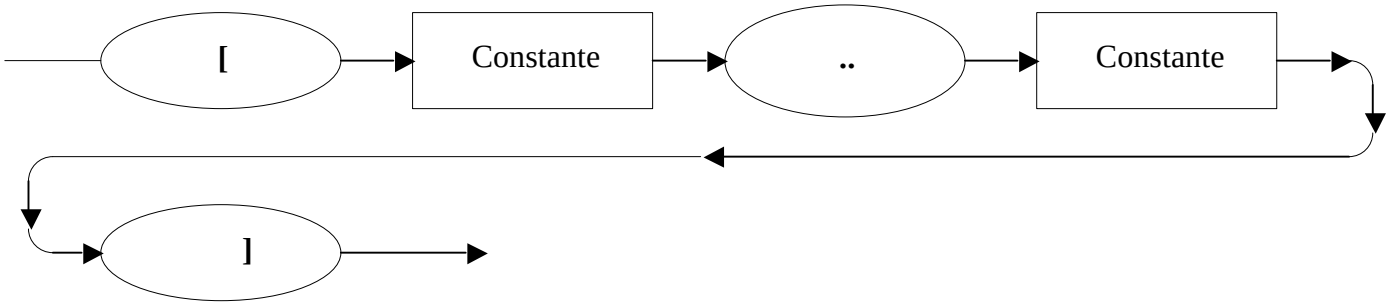
Tipo Básico:



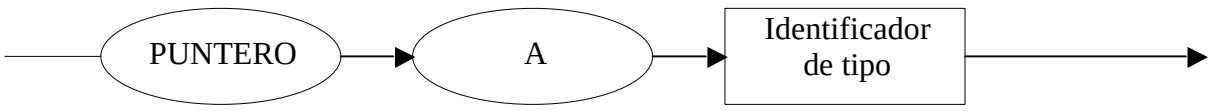
Tipo Enumerado:



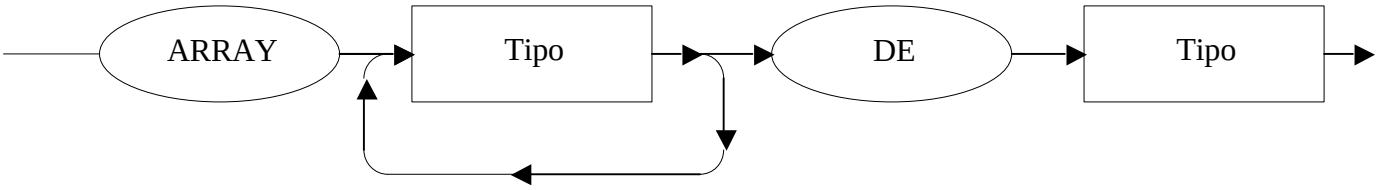
Tipo Subrango:



Tipo Puntero:

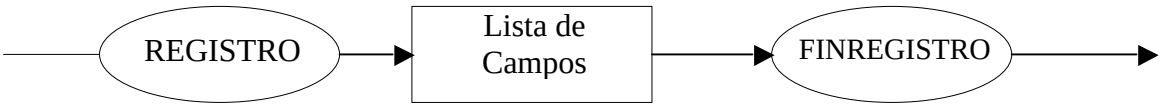


Tipo Array:

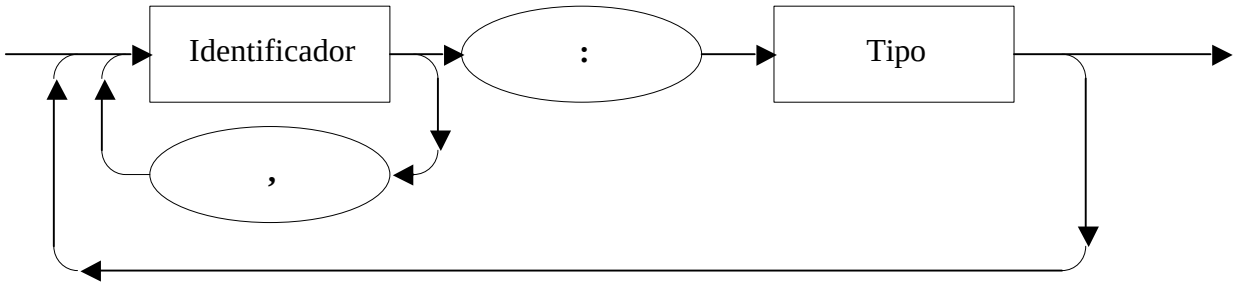


Tipo Fichero:

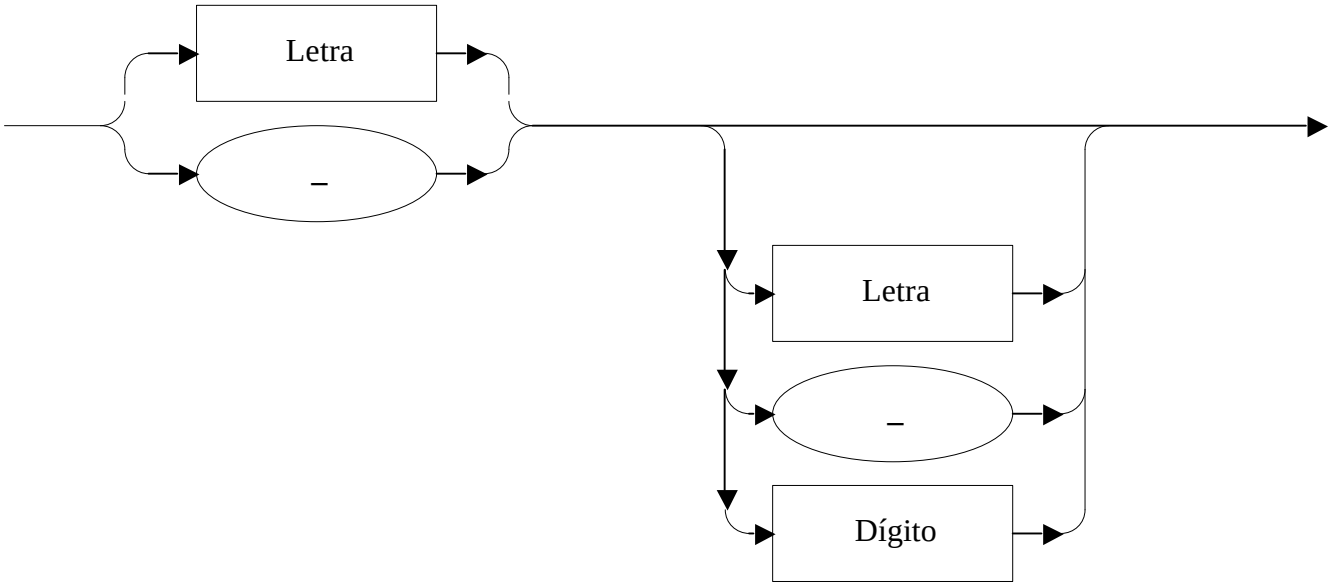
Tipo Registro:



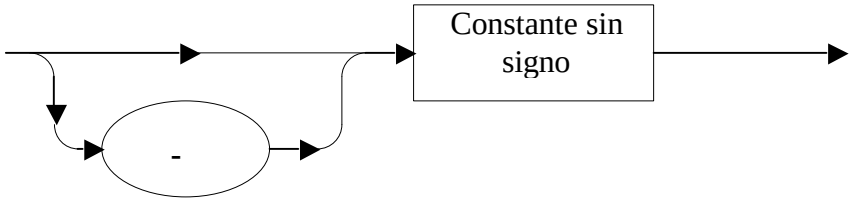
Lista de Campos:



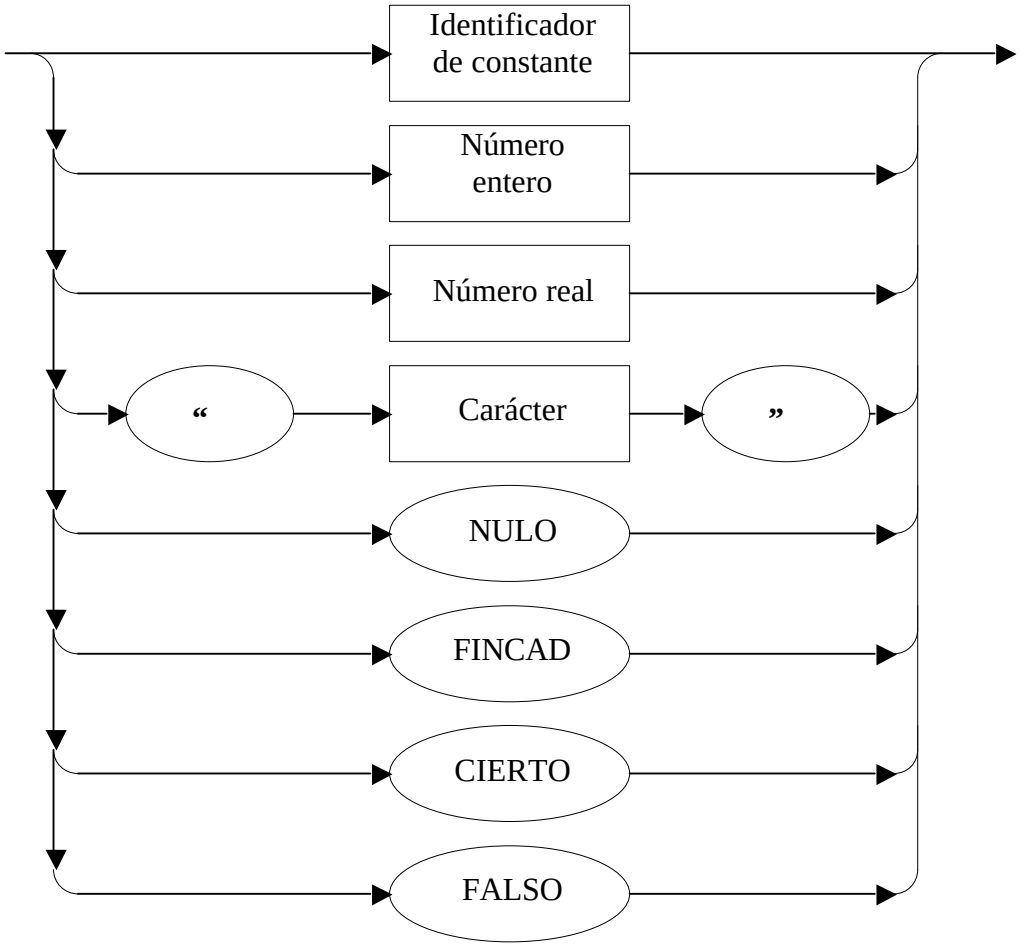
Identificador:



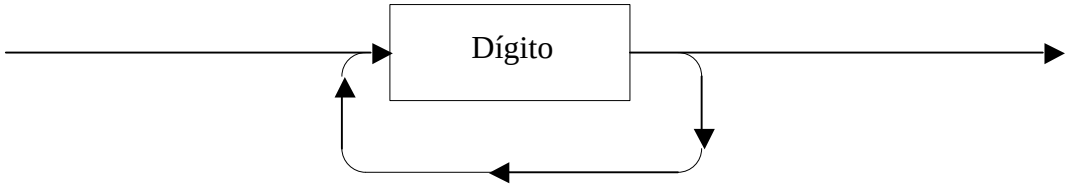
Constante:



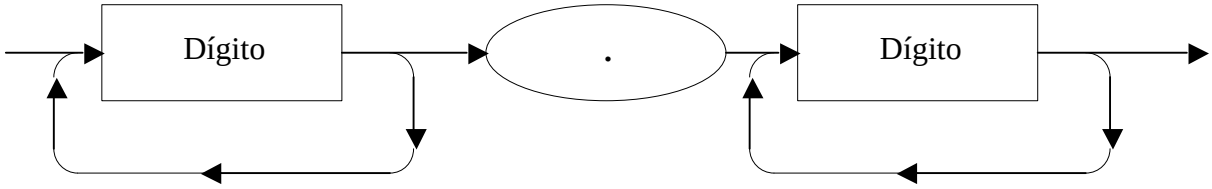
Constante sin signo:



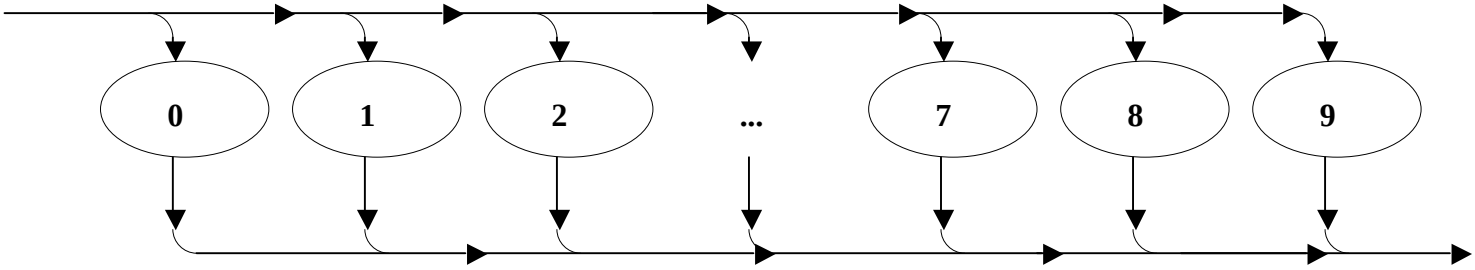
Número entero:



Número real:



Dígito:



Letra: