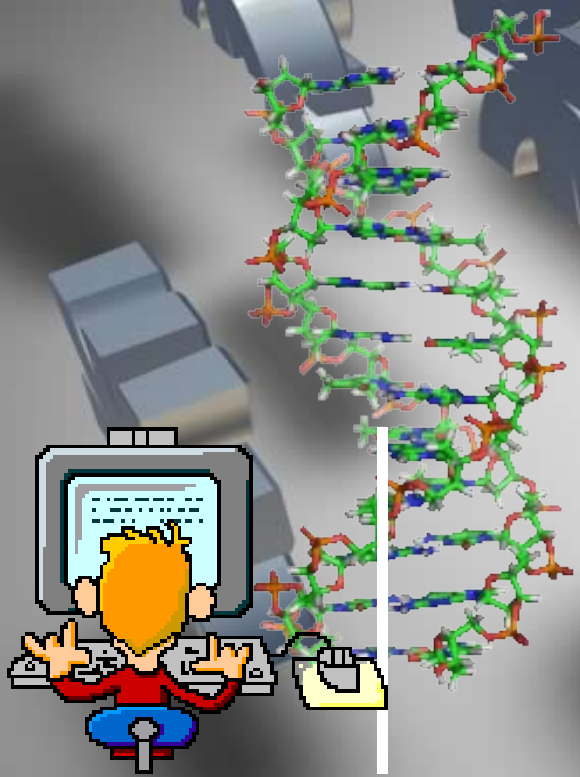
Programación I

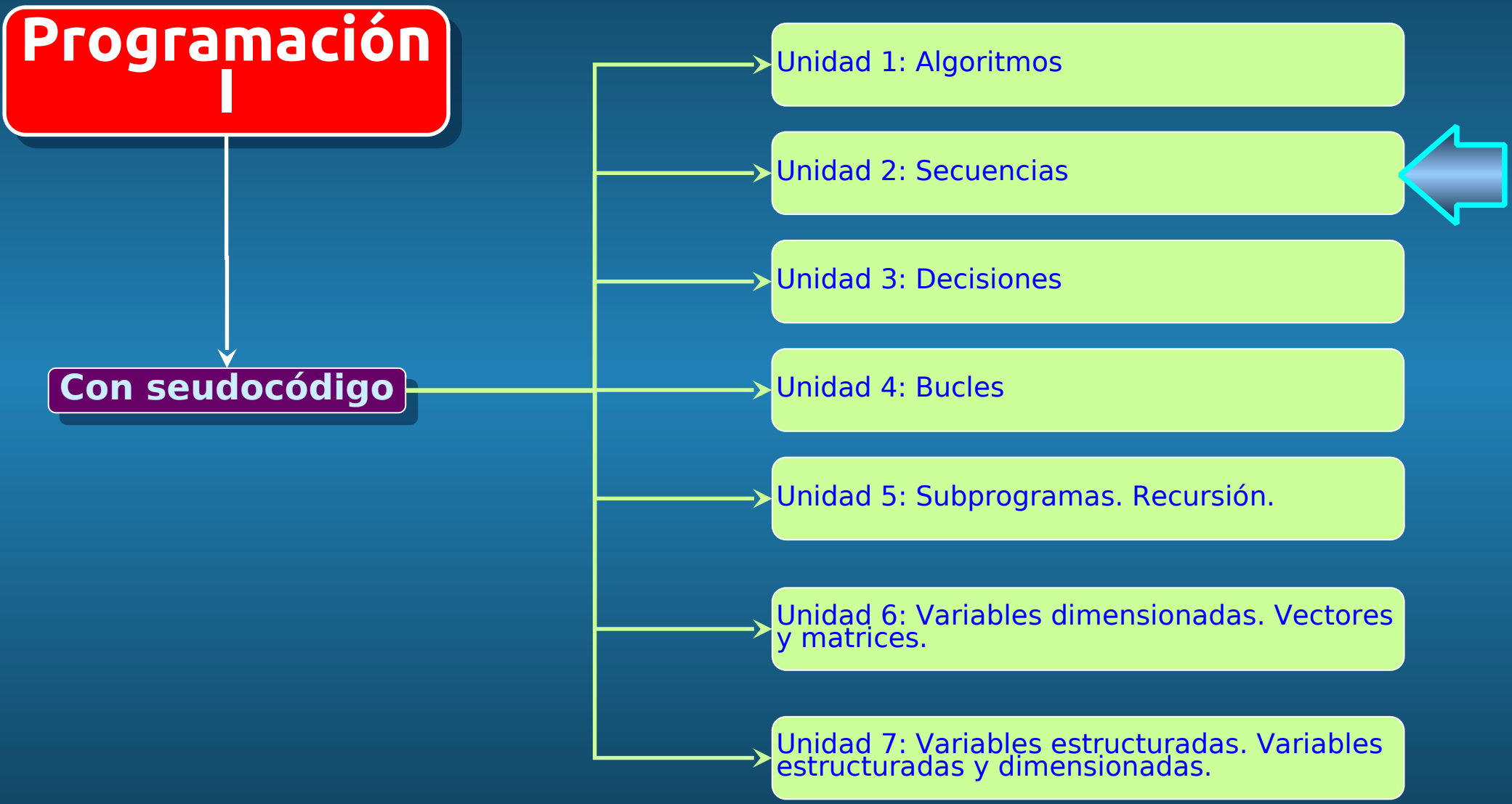
Ing. Carlos R. Rodríguez

Tecnicatura Universitaria en Programación



UNIVERSIDAD TECNOLÒGICA NACIONAL

Facultad Regional Mendoza


Programación
I
Con seudocódigo
Unidad 1: Algoritmos
Unidad 2: Secuencias
Unidad 3: Decisiones
Unidad 4: Bucles
Unidad 5: Subprogramas. Recursión.
Unidad 6: Variables dimensionadas. Vectores y matrices.
Unidad 7: Variables estructuradas. Variables estructuradas y dimensionadas.

Sobre el estilo estructurado:

Teorema Fundamental de la Programación Estructurada:

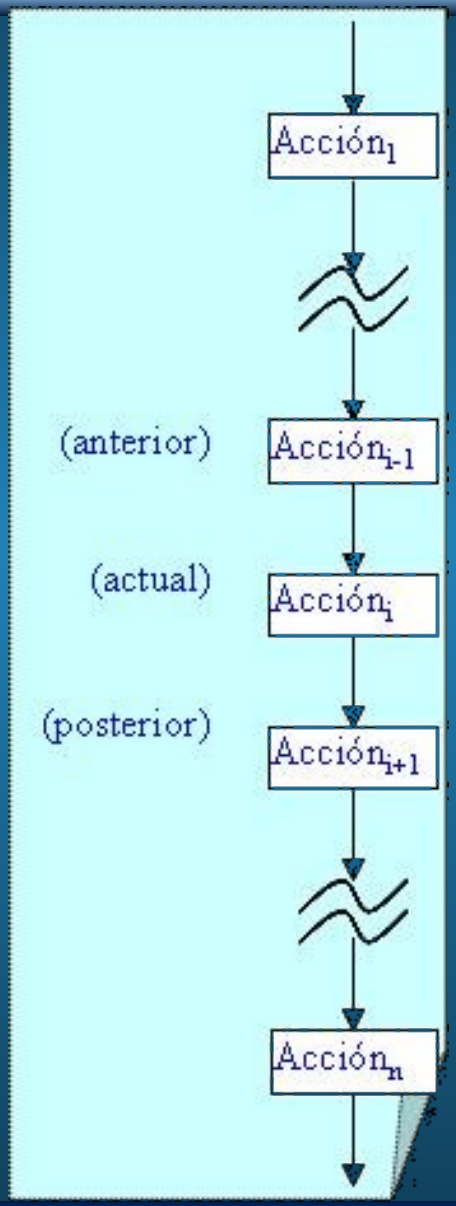
- Todo programa puede elaborarse utilizando solo 3 tipos de instrucciones:
  - **Secuencias**
  - **Decisiones**
  - **Bucles**

Sobre las secuencias:

Una secuencia podría describirse como una **cadena**, donde cada eslabón posee un *eslabón precedente* o *anterior*, y un *eslabón sucedente* o *posterior*.

Esta regla general posee dos *excepciones*:

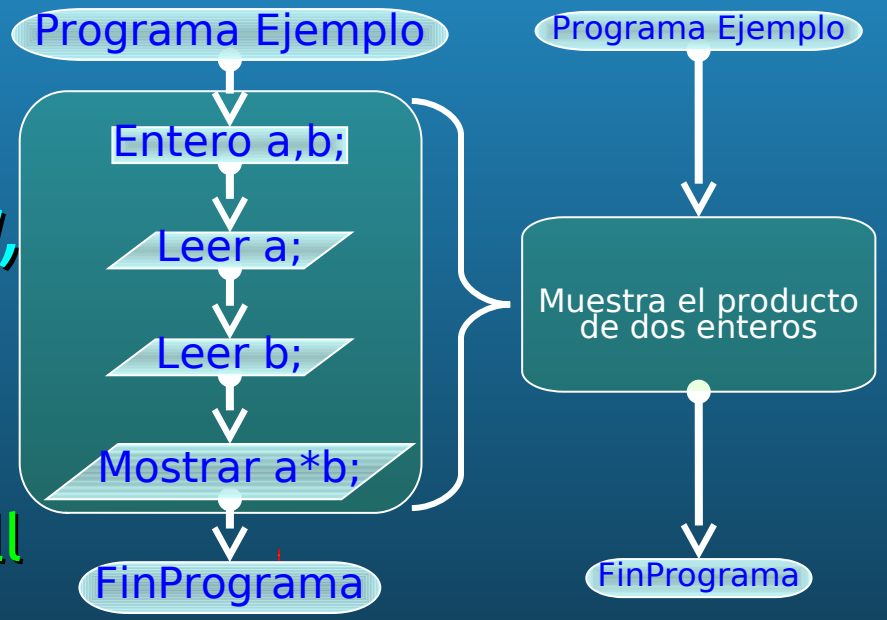
- El **primero** (no tiene *anterior*)
- El **último** (no tiene *posterior*).



Sobre las secuencias...

"(...) *descubrir un* ***proceso*** *es interpretar un acontecimiento como un conjunto de acciones más elementales, cuyo* ***efecto acumulativo es el mismo que el producido por el acontecimiento completo*** (...) *Si una acción de este proceso no puede empezar antes que la acción en curso esté completamente terminada, entonces el proceso se denomina secuencial*".

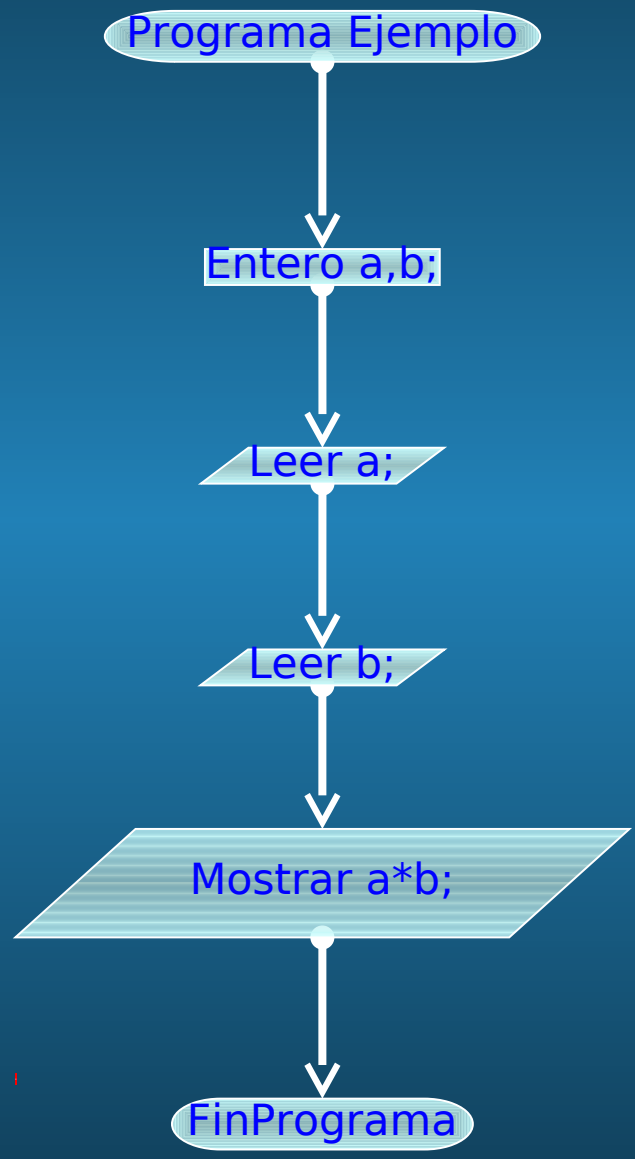
Jèan-Pièrre Peyrin & Pièrre-Clàude Scholl



Sobre las secuencias...

¿Qué acciones son secuenciales?

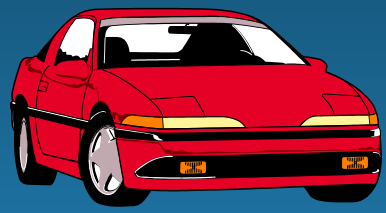
- Declaración de *variables*
- Asignación
- Lectura
- Escritura
- Llamada a *procedimiento*
  - Es una acción *diseñada* por el **programador**

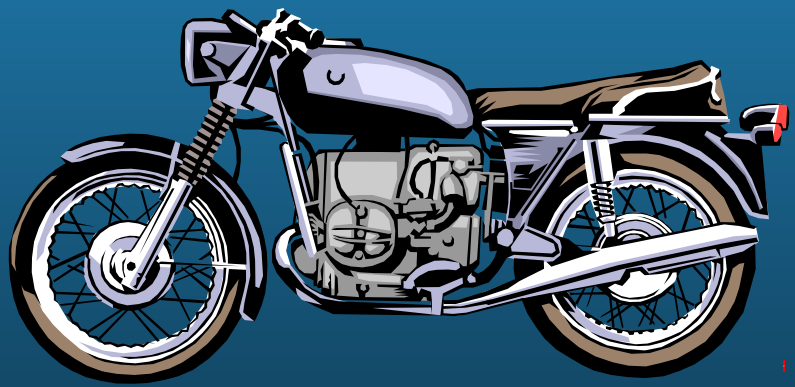


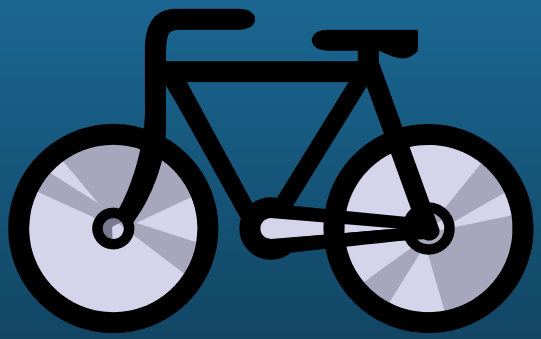
Tipos de datos primitivos:

Usualmente reconocemos distintos tipos de datos de la realidad circundante.

- Así, por ejemplo, normalmente reconocemos (y diferenciamos) un automóvil de una moto y ésta de una bicicleta.

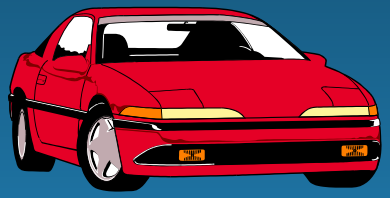


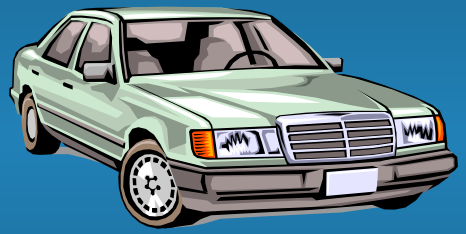


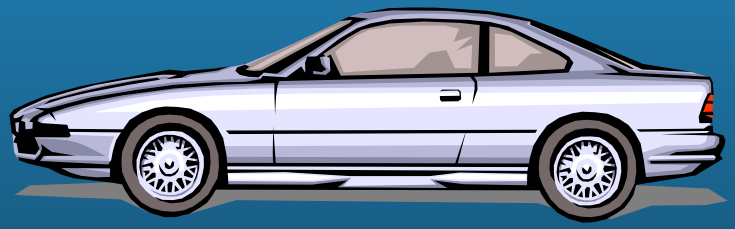


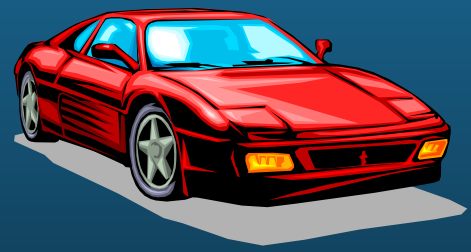
Tipos de datos primitivos...

Pero en otras circunstancias podemos desear agruparlos bajo la común categoría de ***vehículos***, o bien descomponer alguno de ellos, digamos los automóviles según su marca, para una misma marca según su modelo, luego según el color, y así sucesivamente.



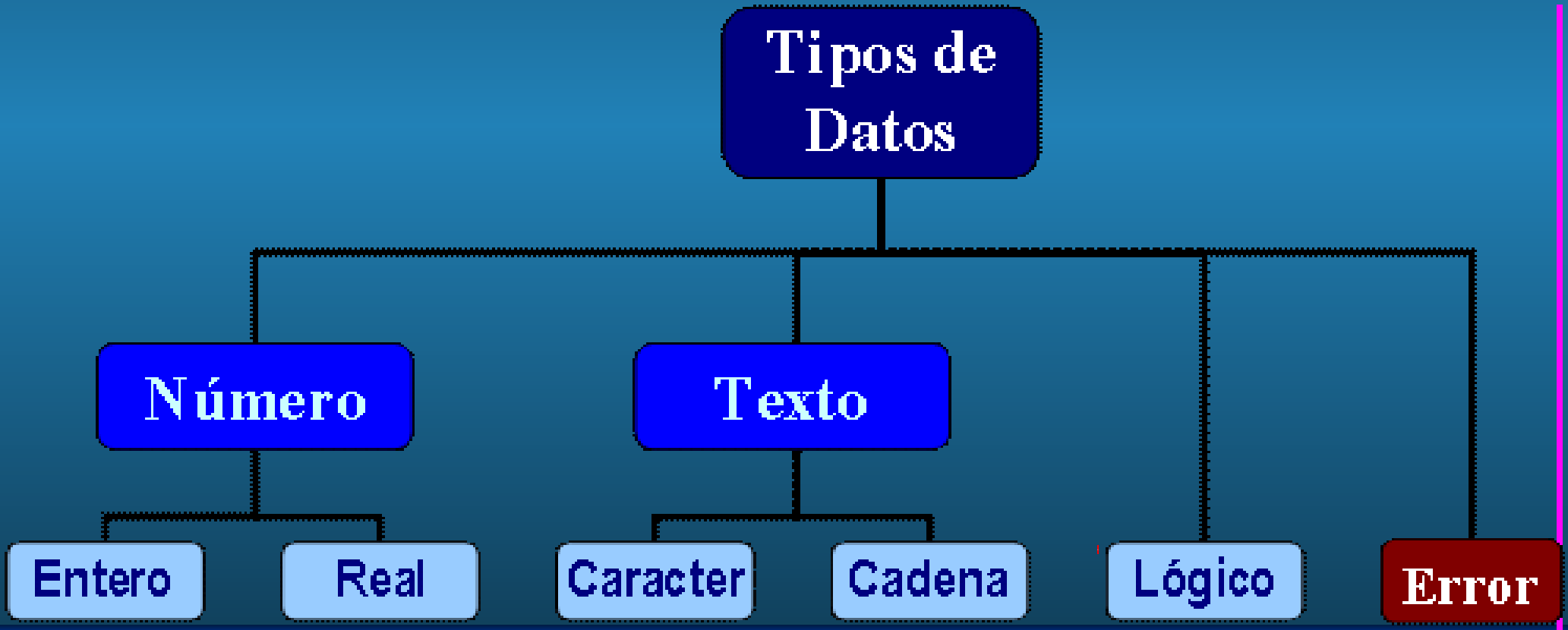






Tipos de datos primitivos...

Determinan el conjunto de valores posibles para constantes, variables o para resultados de expresiones y funciones.
Utilizaremos como tipos primitivos:



Diagramas de Conway

Se usan para expresar sintaxis. Consisten en flechas que marcan el o los recorridos permitidos, pasando por dos tipos de elementos de dibujo:

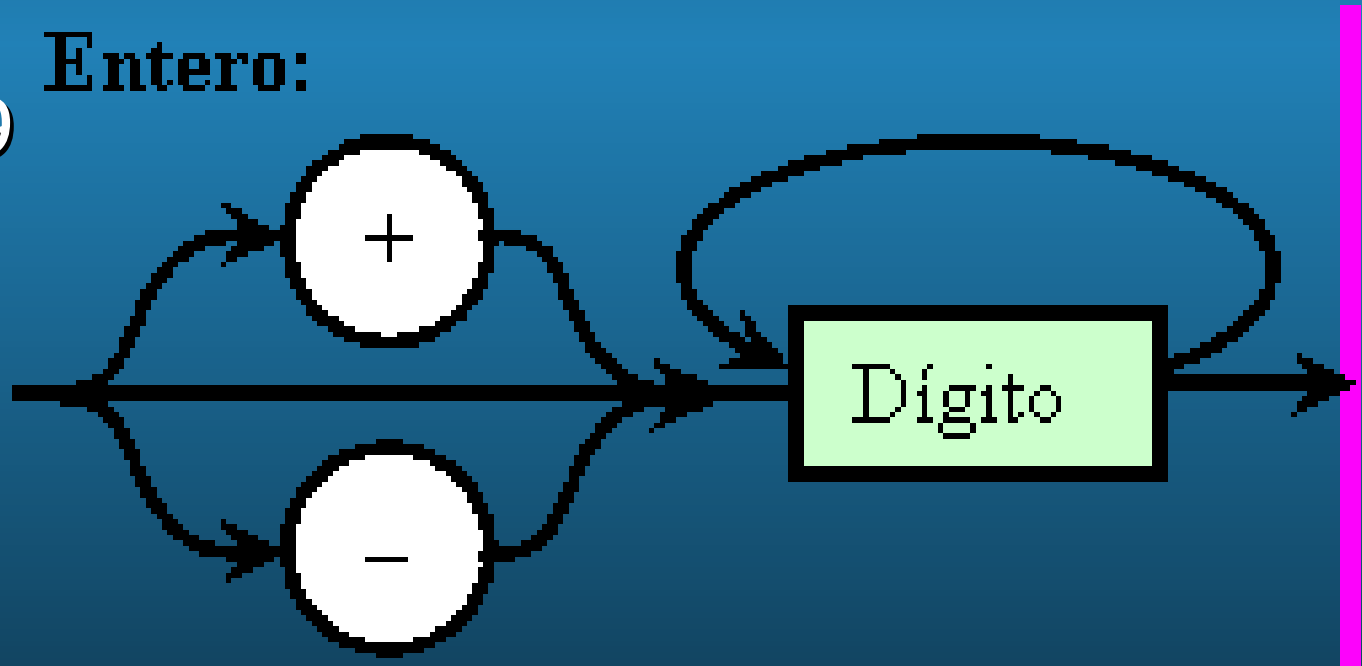
- El **óvalo** o **círculo**, que significa contenido literal.
- El **rectángulo**, que implica un concepto para el que es necesaria una posterior ***descomposición*** *antes de llegar al texto.*

En el ejemplo: cualquier entero positivo

Tipo Numérico Entero

Un valor entero se construye con un signo opcional positivo o negativo al comienzo y una secuencia de uno o más dígitos.

- Ejemplos:
  - -123456789
  - +123456789
  - 123456789



Tipo Numérico Entero...

Nótese que esto implica que no hay **límites** a la cantidad de dígitos, sino que se deben utilizar todos los ***necesarios***.

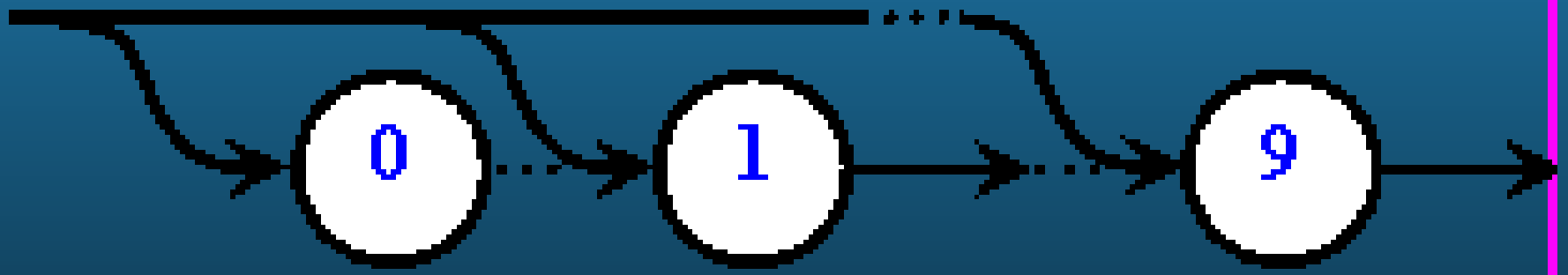
- En este planteo, un valor como **un billón** será formado por un uno *seguido de doce ceros*. Y será entero.
- Esta es una característica que buena cantidad de lenguajes hoy no soportan, pero que es **conveniente** para acercar el problema a nuestro pensamiento *sin condicionarlo previamente por el contexto de ejecución*.

Dígito

El concepto dígito necesita su propia regla sintáctica, que nos permita **escribir** *cualquier* dígito con base diez (*abandonando casi definitivamente toda pretensión de acercarnos a los números binarios presentes en toda computadora de nuestra época*).

- Esto abarca desde el cero al nueve, inclusive:

Dígito:



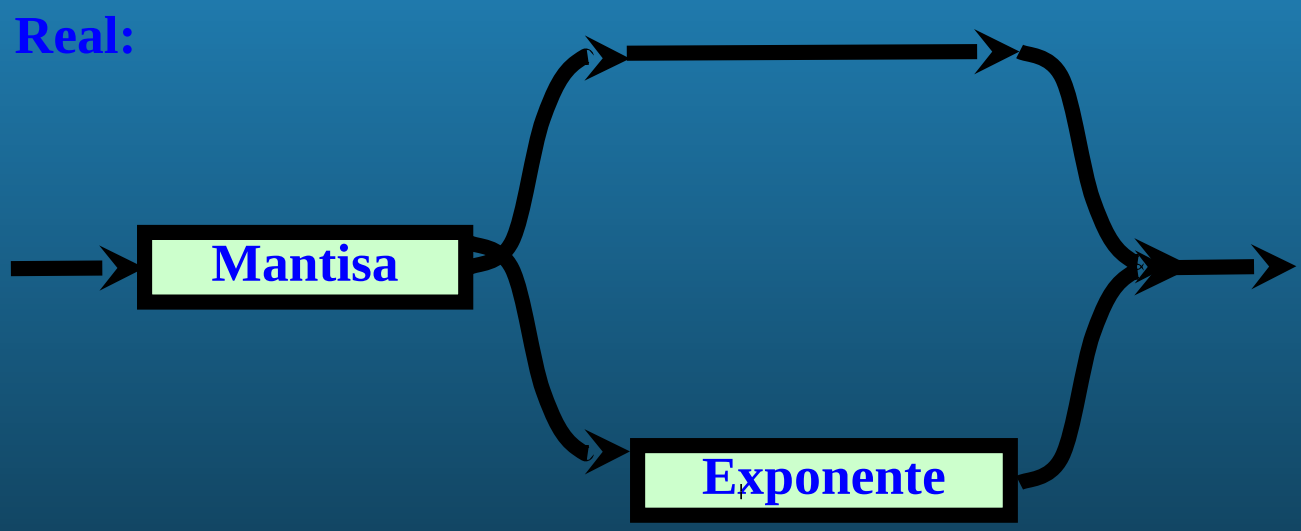
Tipo Numérico Real

Los números reales o de punto flotante deben admitir valores ***muy pequeños*** y ***muy grandes***.

Además deben ser capaces de utilizar la ***notación científica***, por ejemplo: **1.234E+2** (*123.4*)

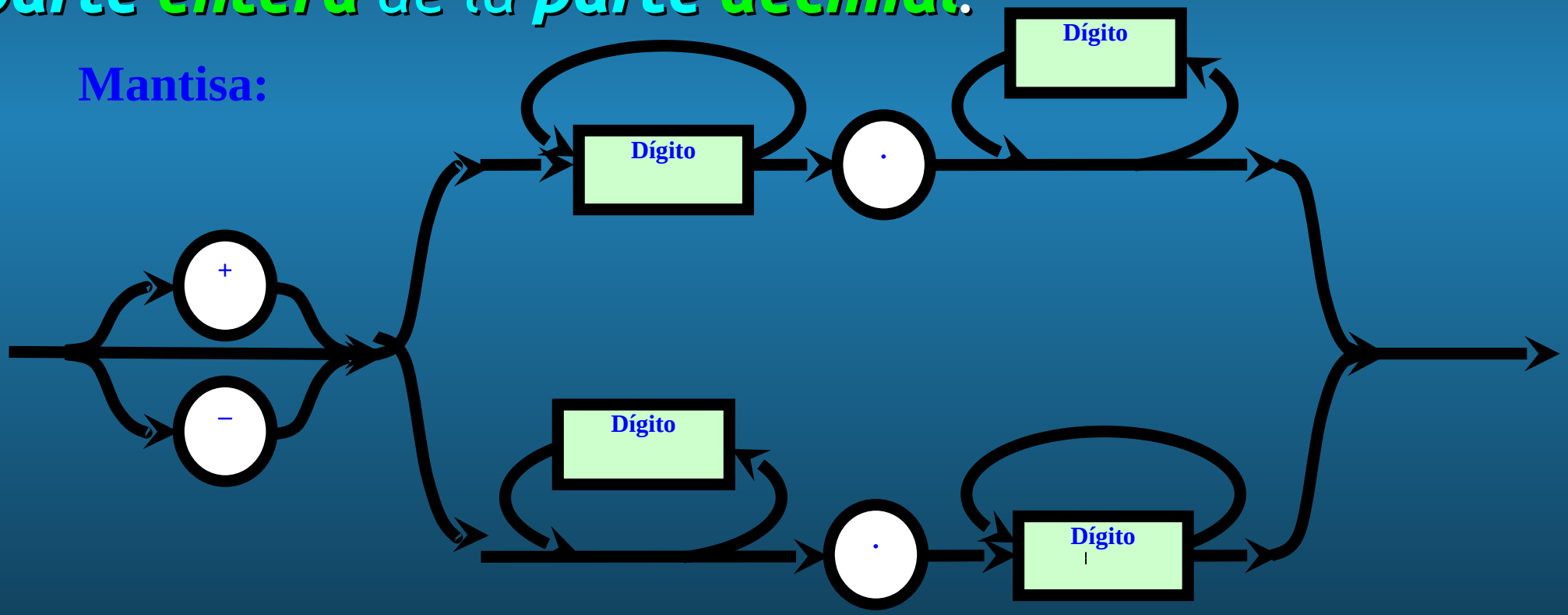
Está compuesto por dos partes:

- Mantisa
- Exponente.



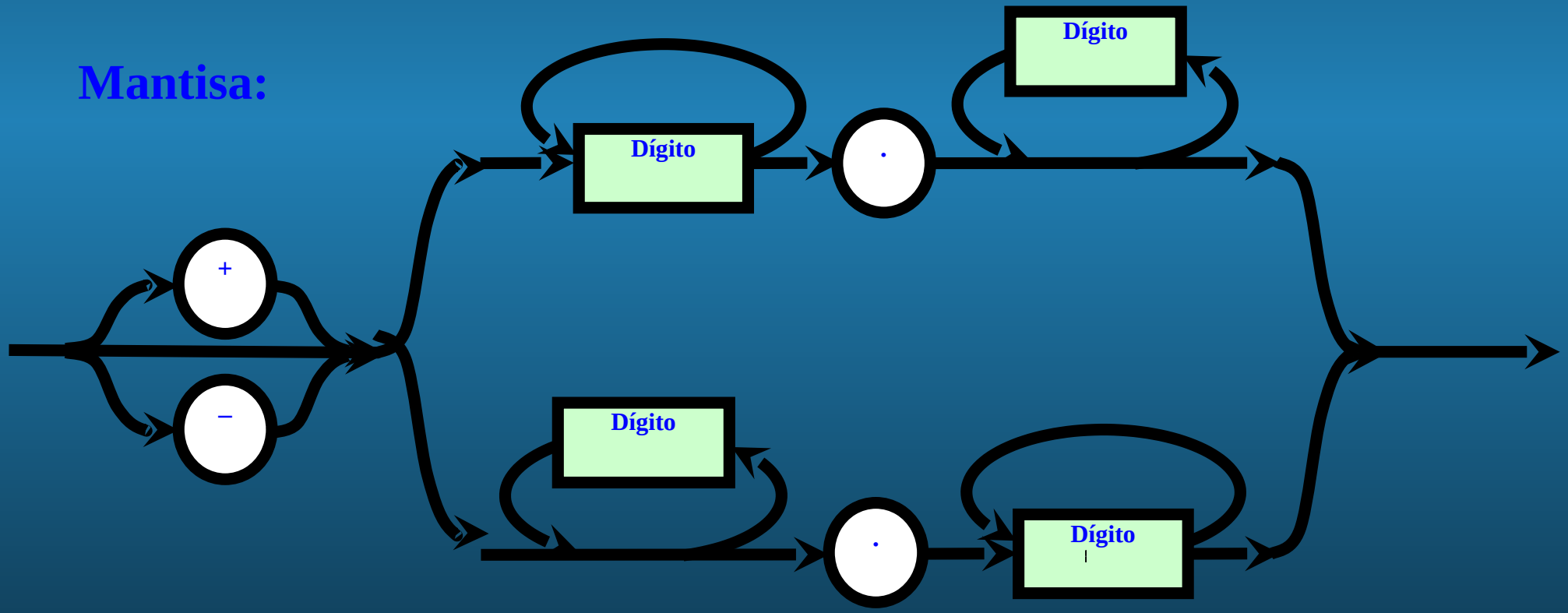
Mantisa

Está compuesta por un signo (*positivo* o *negativo*) *opcional* y luego posee una secuencia de ***uno o más dígitos*** con un ***punto intercalado*** que ***separa*** la ***parte entera*** de la ***parte decimal***.



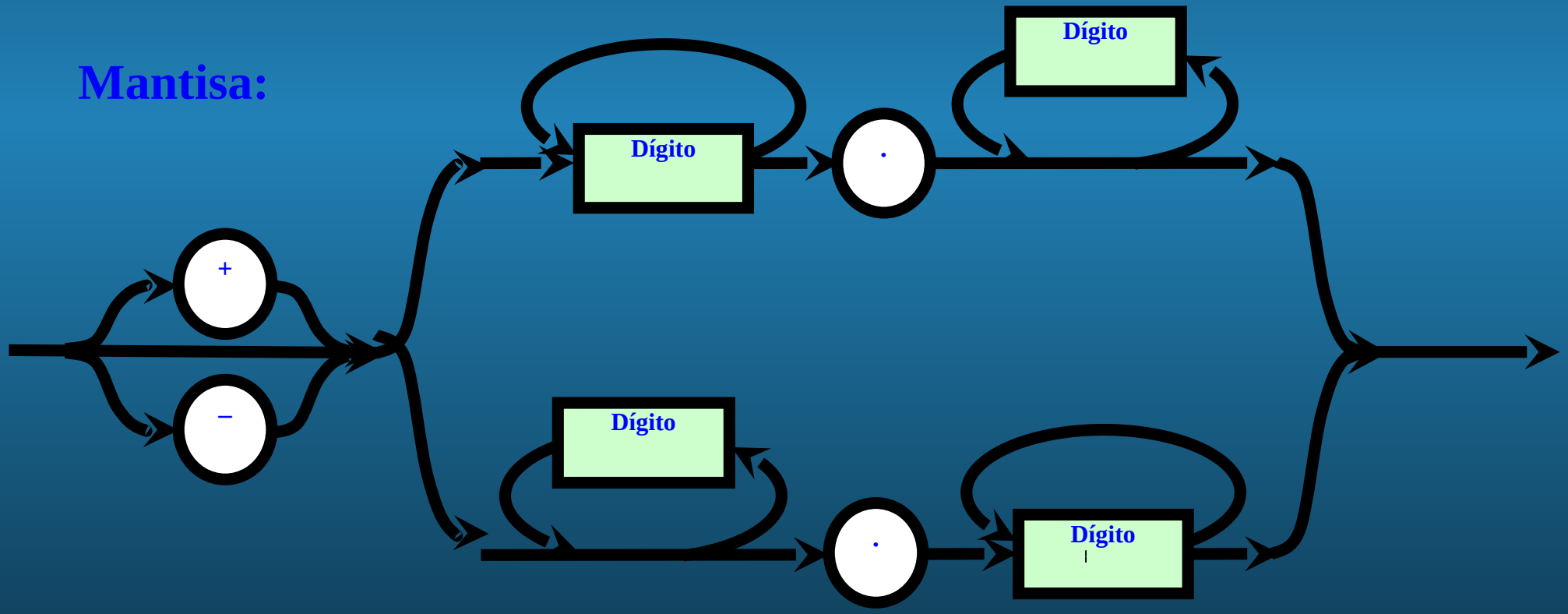
Mantisa...

Puede contener cualquier **cantidad *positiva*** de dígitos ***significativos*** y el punto puede estar en ***cualquier posición dentro de la mantisa***.



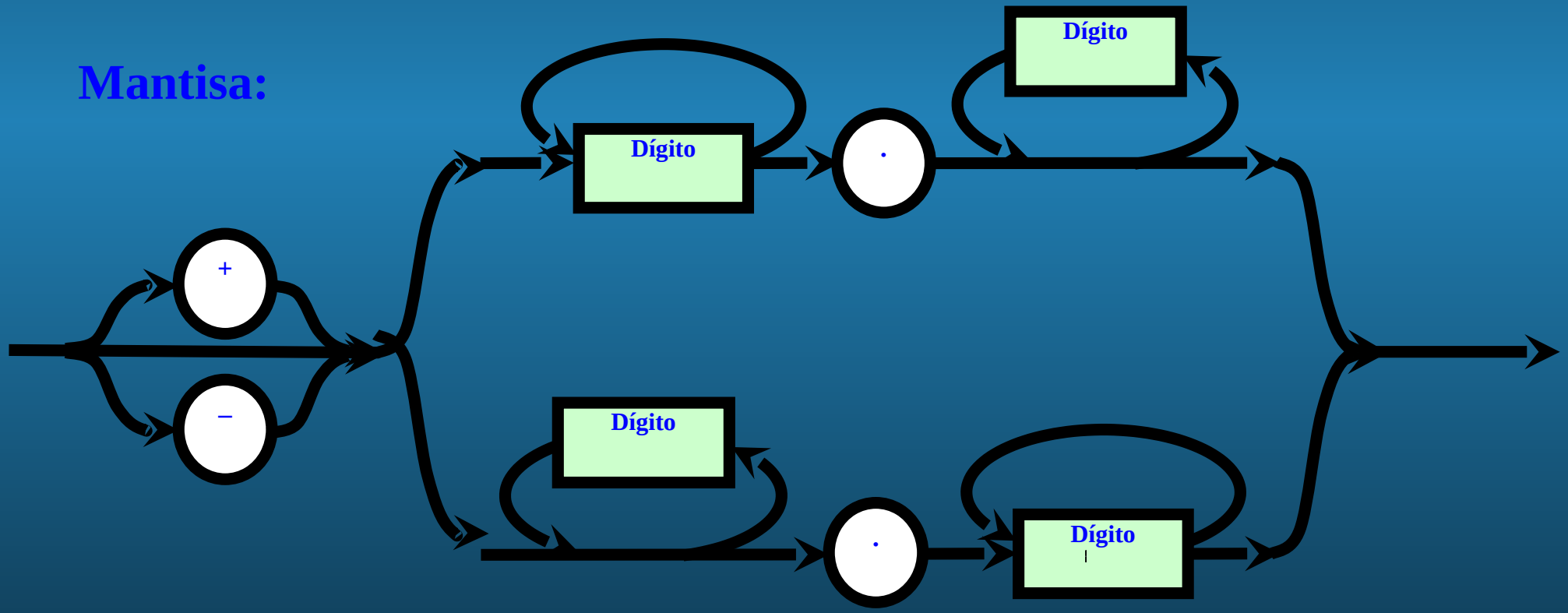
Mantisa...

Así, si no hay dígitos ***antes*** del punto decimal, **al menos uno será *obligatorio después*** del punto.



Mantisa...

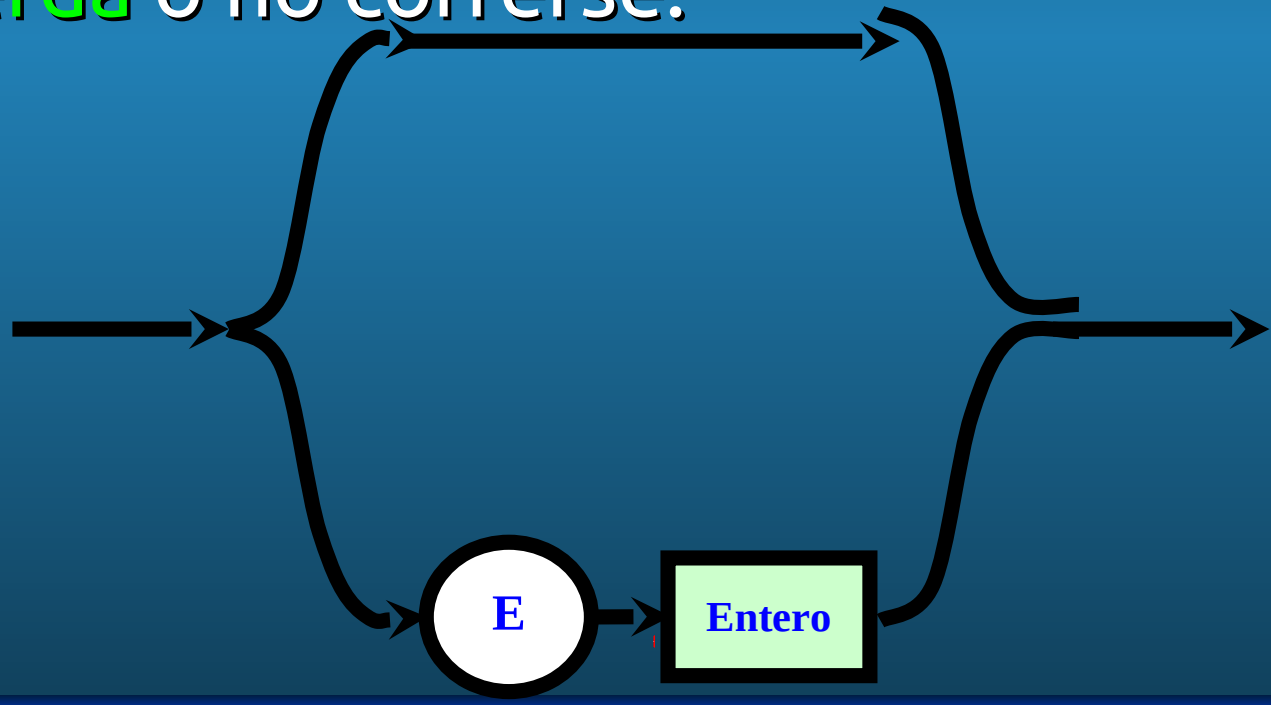
Inversamente, si hubiera al menos un dígito ***antes*** del punto, será ***opcional*** que existan dígitos ***luego de él***.



Exponente

Es un valor entero precedido de una E ***mayúscula*** o ***minúscula***:

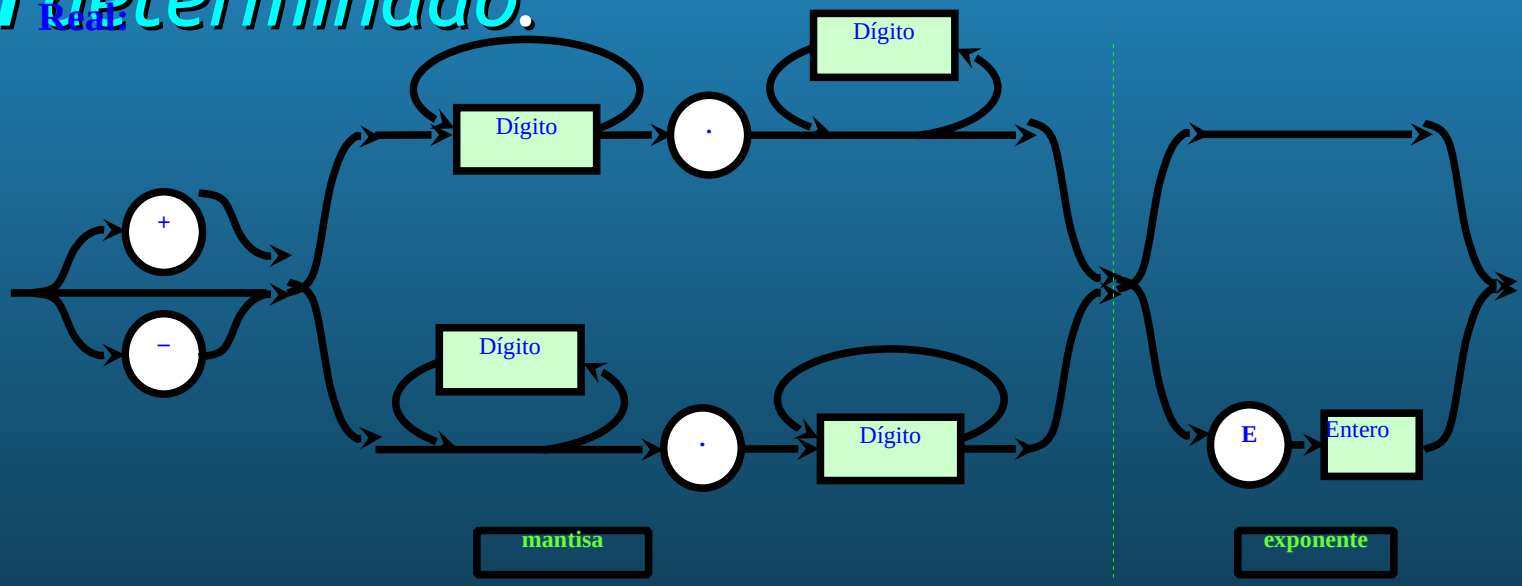
- El valor puede ser positivo, negativo o cero, indicando si el punto decimal debe correrse a derecha o izquierda o no correrse.



Tipo Real (conclusión)

Un número real así poseerá precisión y rango ***tan grandes como fuera necesario***, como consecuencia de la inexistencia de límites a la cantidad de dígitos.

Esto, obviamente, *necesitará cierto cuidado a la hora de la **implementación en un lenguaje de programación** determinado*.

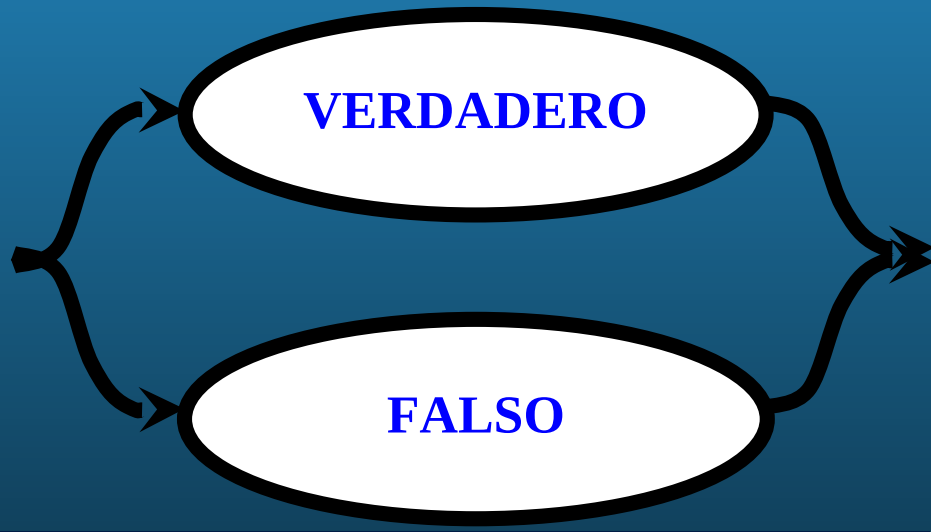


Tipo Lógico

En este caso sólo se trata de dar cabida a los valores verdadero y falso.

- Para ello se utilizarán (*y sólo para eso*) los identificadores VERDADERO y FALSO, que de ahora en más pasan a ser palabras *reservadas*.

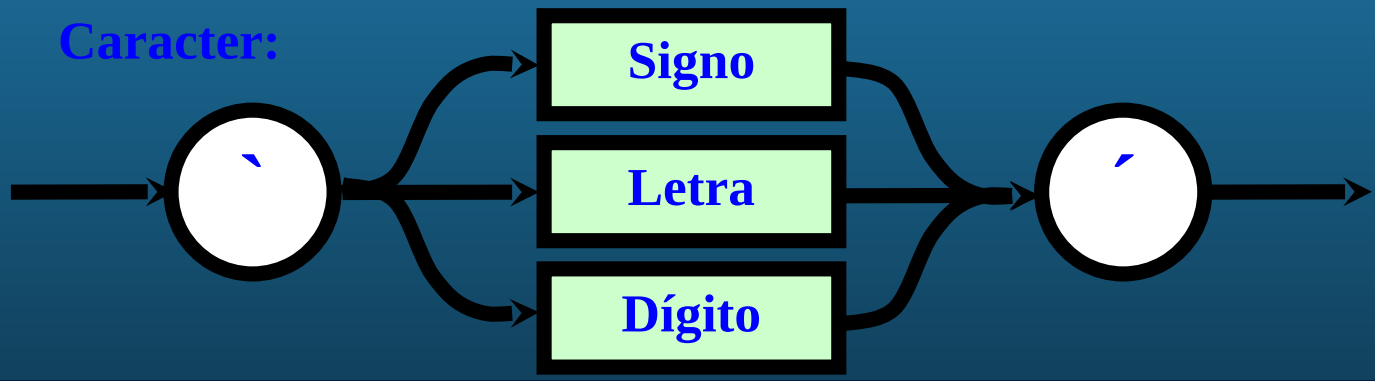
Lógico:



Tipo Textual Caracter

Este tipo de datos hace referencia a un texto de un único carácter y es *ampliamente aceptado* dentro de los **lenguajes de programación** vigentes.

- Está delimitado por **apóstrofes**.



Letra

Es toda letra de nuestro alfabeto, ***minúscula*** o ***mayúscula***.

- Eso incluye:
  - Vocales acentuadas.
  - U con diéresis.
  - Eñe.
- ***Algunas*** letras se utilizan para construir **identificadores**.

a b c d e f g h i j k l m n ñ o p q r s t u v w x y z
A B C D E F G H I J K L M N Ñ O P Q R S T U V W X Y Z
á é í ó ú Á É Í Ó Ú ü Ü

Signo

Denominaremos signo a todo texto que **no sea letra ni dígito ni apóstrofe ni comillas** (*definición por **exclusión***).

- No forman parte de las palabras.
  - Por lo tanto no forman parte de los identificadores.
- Ejemplos:

, . ; : ( ) [ ] { } ¿ ? ¡ ! $ & / + * - =

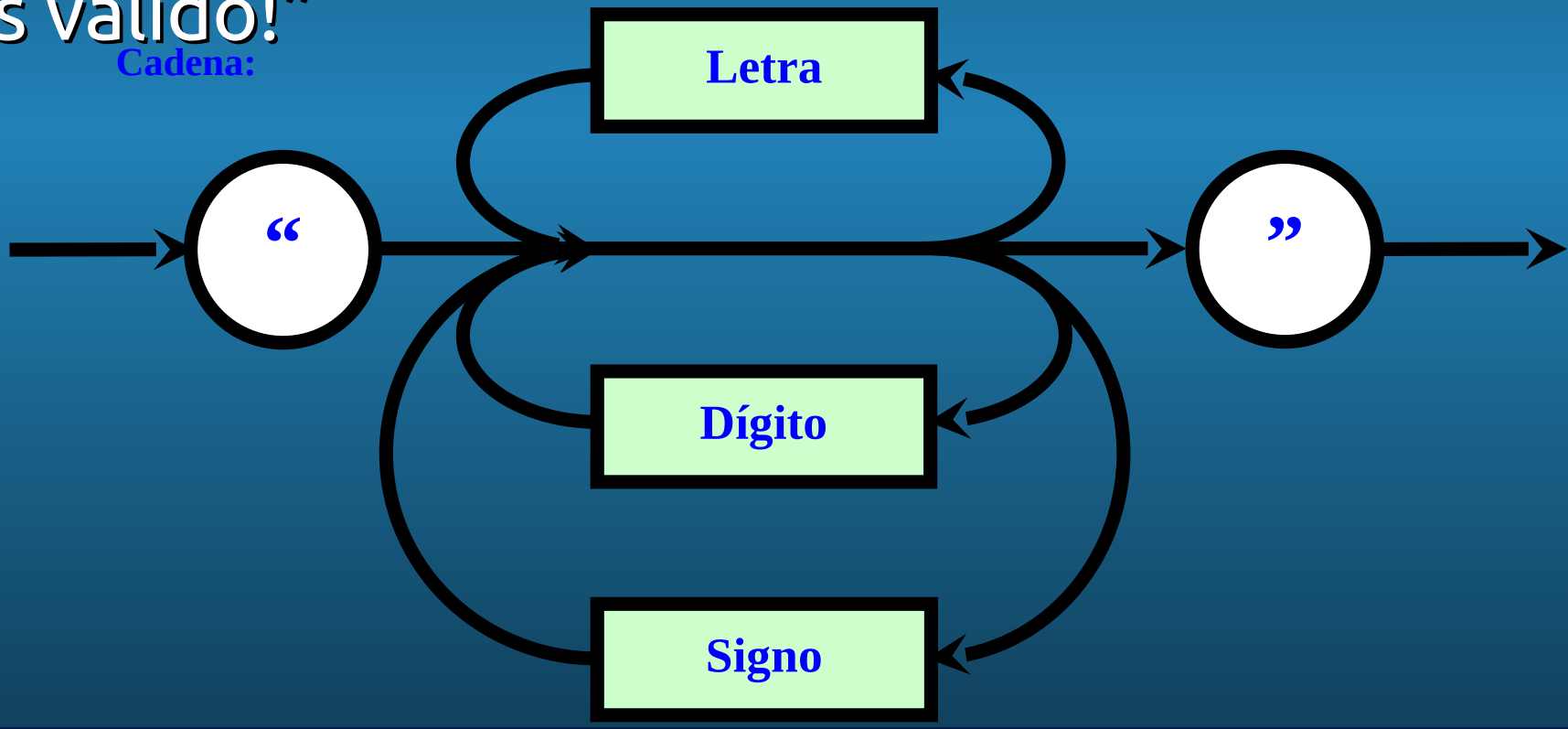
Tipo Textual Cadena

Es un texto de cero o más caracteres delimitados por comillas. Por su naturaleza variable suele tener una de las siguientes implementaciones:

- Longitud máxima fija.
  - Asignación fija de memoria automática.
    - Pascal.
- Longitud variable.
  - Asignación dinámica de memoria.
    - C/C++.

Tipo Textual Cadena...

- Ejemplos:
  - ""
  - "R2D2"
  - "¡Es válido!"



Tipo Error

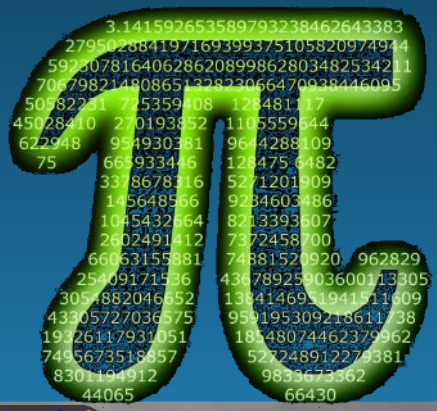
Un valor *erróneo* - *desde el punto de vista* ***sintáctico*** - es aquél que no puede ser construido utilizando alguno de los diagramas anteriores y por tanto no pertenece a alguno de ellos.

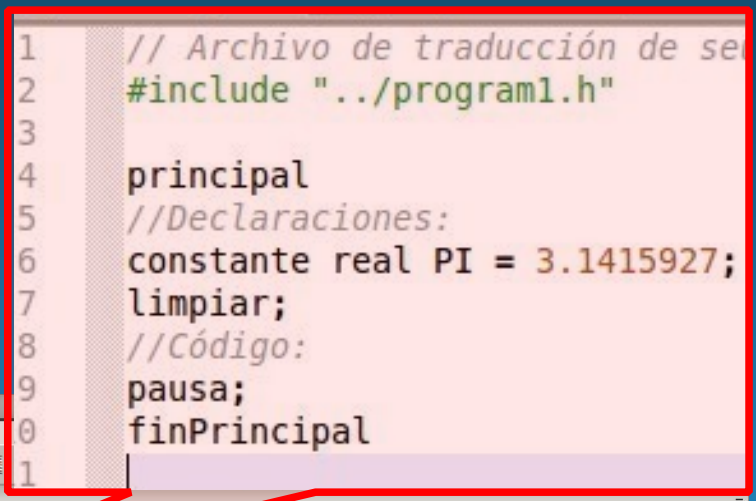
Es una definición por **descarte**: todo lo que no es válido es erróneo.

Los errores producen errores dondequiera que aparezcan.

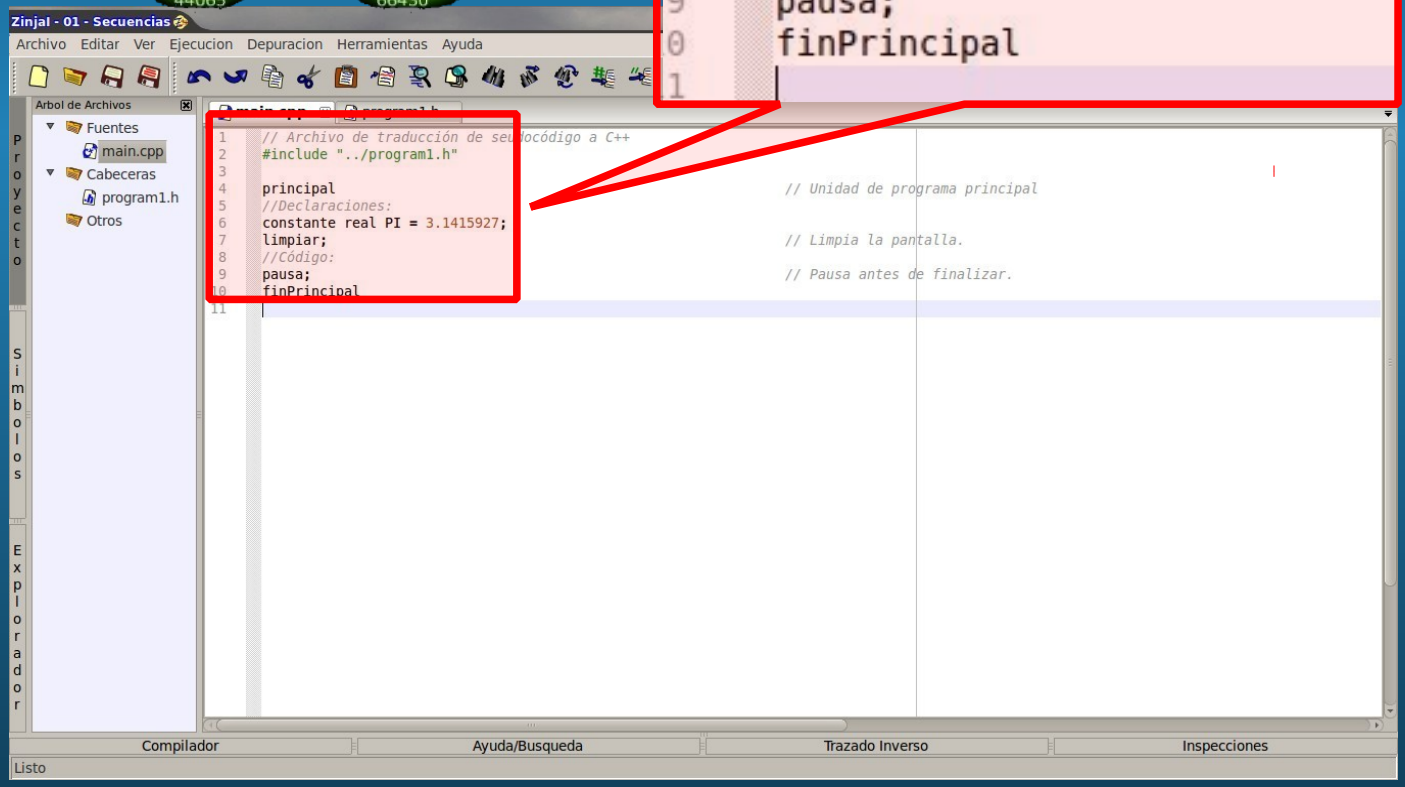
Constantes

Cuando un valor está **incrustado** *dentro del código*, queda claro por su propia naturaleza que – *una vez* ***compilado*** – **no va a cambiar**, es decir, es constante.



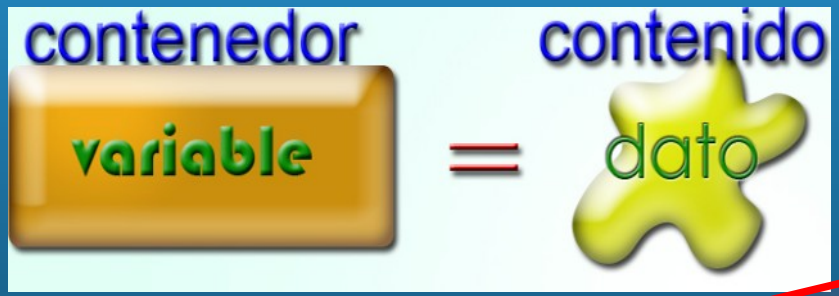


```
1   // Archivo de traducción de se
2   #include "../program1.h"
3
4   principal
5   //Declaraciones:
6   constante real PI = 3.1415927;
7   limpiar;
8   //Código:
9   pausa;
0   finPrincipal
1
```

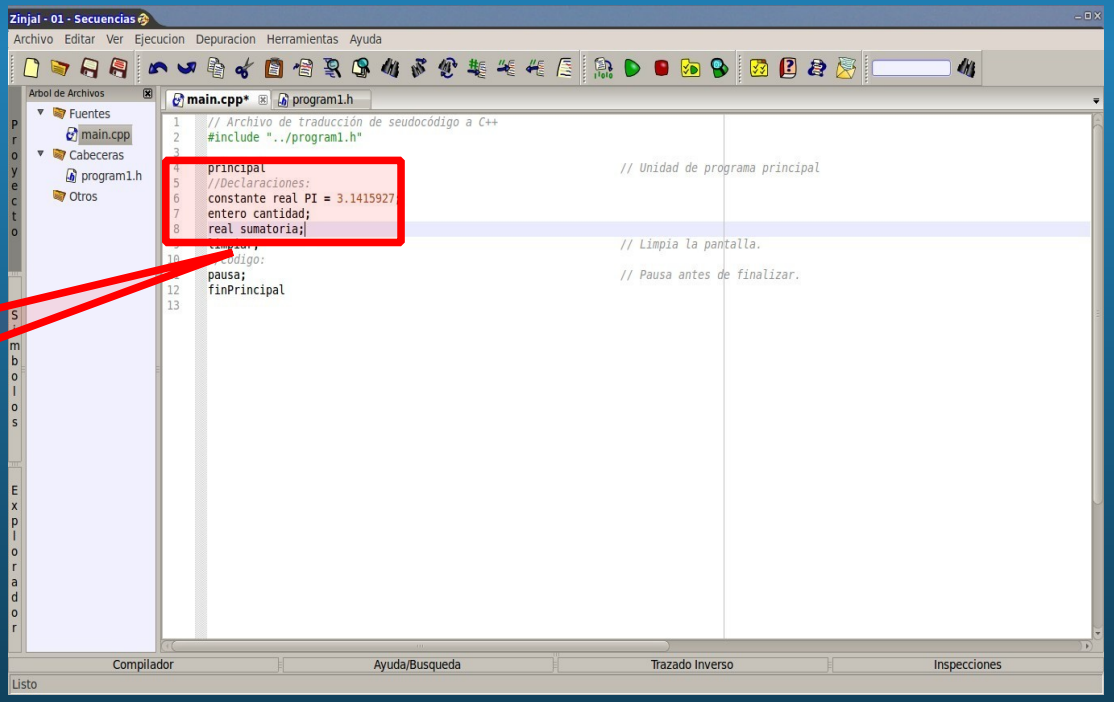


Variables

Si bien las constantes son muy útiles, **es *imprescindible* utilizar valores que *sí cambien durante la ejecución* del programa**, es decir, que sean variables.



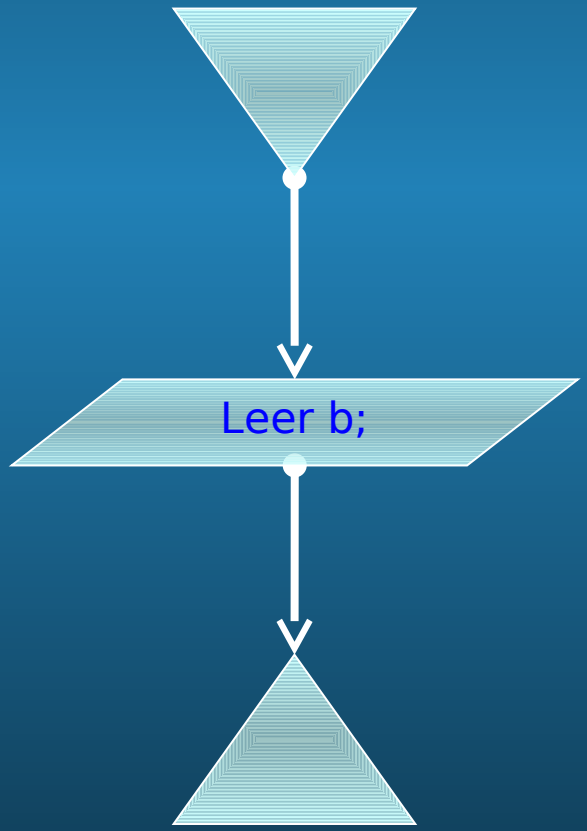

```
5    //Declaraciones:
6    constante real PI = 3.1415927
7    entero cantidad;
8    real sumatoria;
9    limpiar;
```

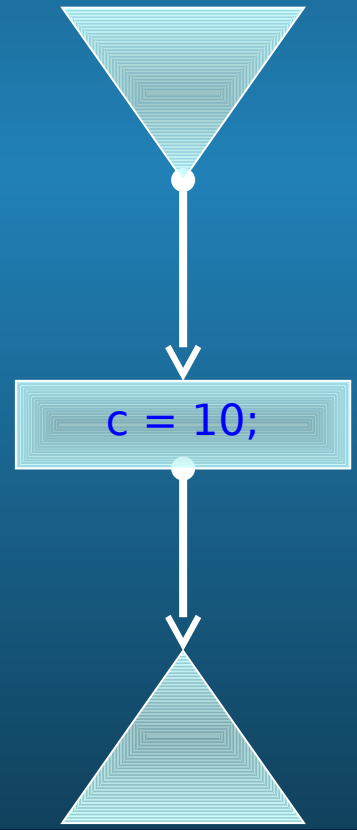


Variables...

Una variable permite - entonces - almacenar un valor. Este puede ser definido (inicializado) mediante alguna acción durante la ejecución del programa:

- Asignación
- Lectura

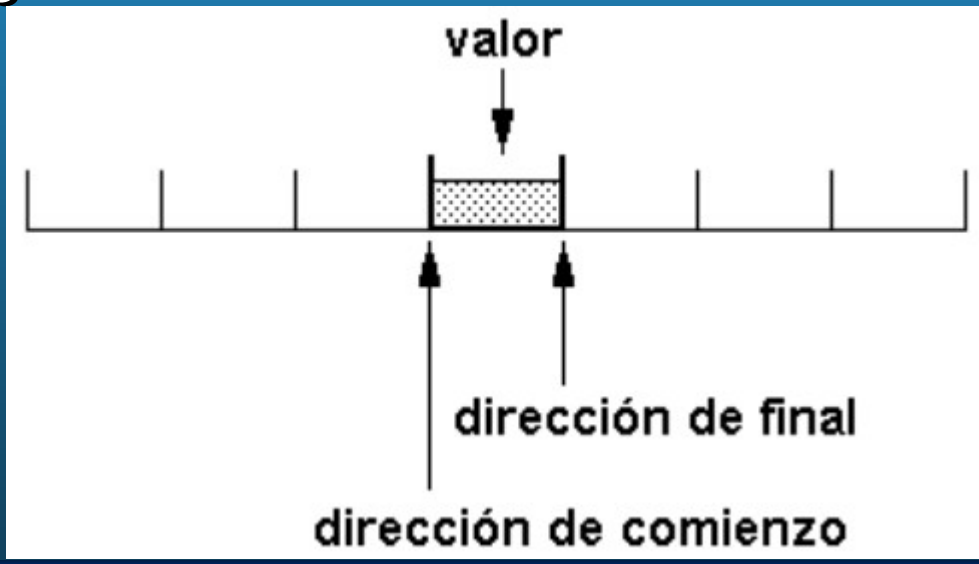




Variables...

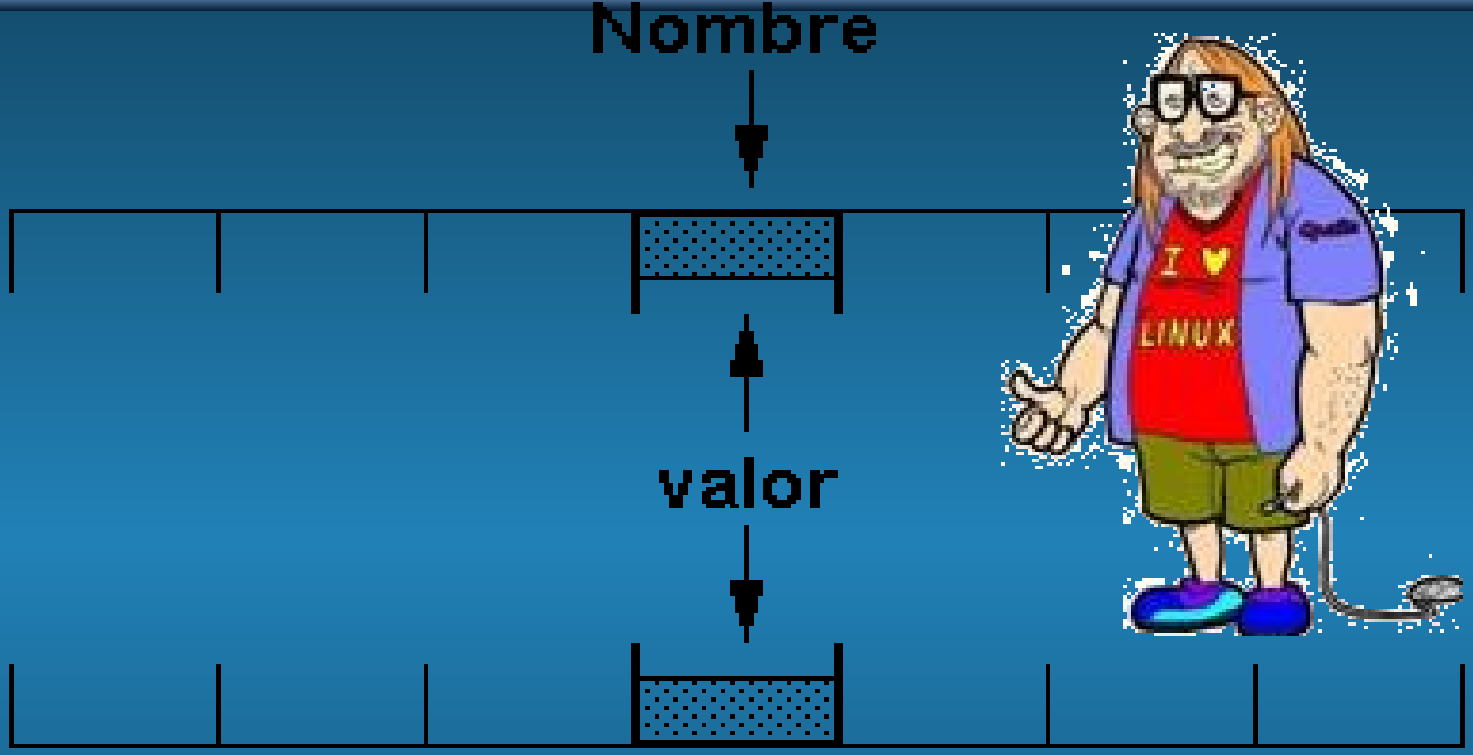
Un programa reconoce a la variable como *un conjunto de posiciones de memoria que expresan un valor.*

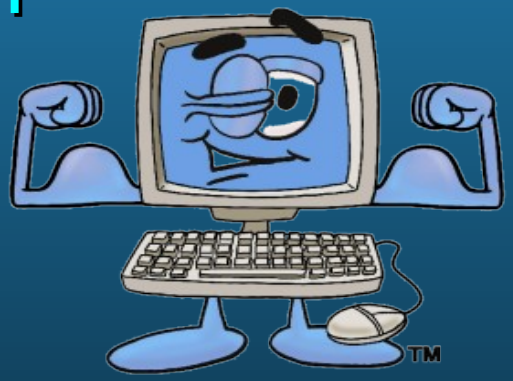
Esta **definición** *difícilmente sea de alguna utilidad para nosotros*, si pretendemos cierto nivel de abstracción respecto de la ejecución.



Variables...

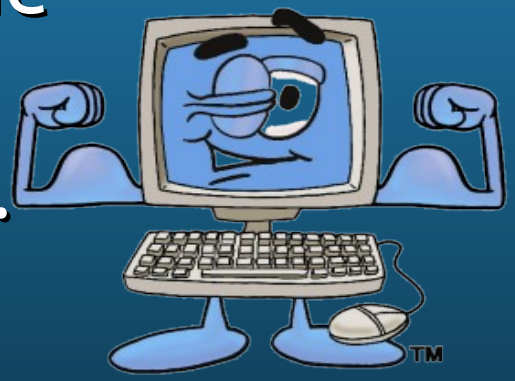
Más conveniente será definir la noción de variable como un único **valor** *asociado* con un único **nombre**.





Variables...

Desde nuestro punto de vista sólo es importante acceder al valor usando el nombre de su contenedor.



Variables...

El nombre entonces será un identificador del **valor *almacenado*** y **podemos usarlo para acceder a éste**.

Deberemos tener en cuenta algunas cosas:

- Cumple las mismas reglas sintácticas de escritura que cualquier otro **identificador**.

Exámenes

Correos

Facturas

Variables...

Deberemos tener en cuenta algunas cosas:

- El nombre identifica a una única variable, es decir, ***no pueden existir dos variables de igual nombre en la misma unidad de programa***.

Variables...

El nombre de una variable no debería prestarse a confusión con otros nombres, por lo cual es conveniente que **el nombre signifique algo al programador** que la utiliza.

```
/*      GNOT General Public License!
    (c) 1995-2007 Microsoft Corporation  */
#include "dos.h"
#include "win95.h"
#include "win98.h"
#include "sco_unix.h"
class WindowsVista extends WindowsXP implements Nothing{}
int totalNewFeatures = 3;
int totalWorkingNewFeatures = 0;
float numberOfBugs = 345889E+08;
boolean readyForRelease = FALSE;
void main {
    while (!CRASHED) {
    if (first_time_install) {
        if ((installedRAM < 2GB) ||
            (processorSpeed < 4GHz)){
                MessageBox("Hardware incompatibity error.");
                GetKeyPress();
                BSOD();}}
        Make10GBswapfile():
        SearchAndDestroy(FIREFOX|OPENOFFICEORG|ANYTHING_GOOGLE);
        AddRandomDriver();
        MessageBox("Driver incompatibily error.");
        GetKeyPress();
        BSOD(); }
//printf("Welcome to Windows 2000");
//printf("Welcome to Windows XP");
printf("Welcome to Windows Vista");
if (still not crashed){
    CheckUserLicense();
    DoubleCheckUserLicense();
    TripleCheckUserLicense();
    RelayUserDetailsToRedmond();
    DisplayFancyGraphics();
    FlickerLED(hard_drive);
    RunWindowsXP();
    return LotsMoreMoney;}
```

Variables...

Al tener almacenado un valor, ***éste deberá ser del mismo tipo que fue declarada la variable***.

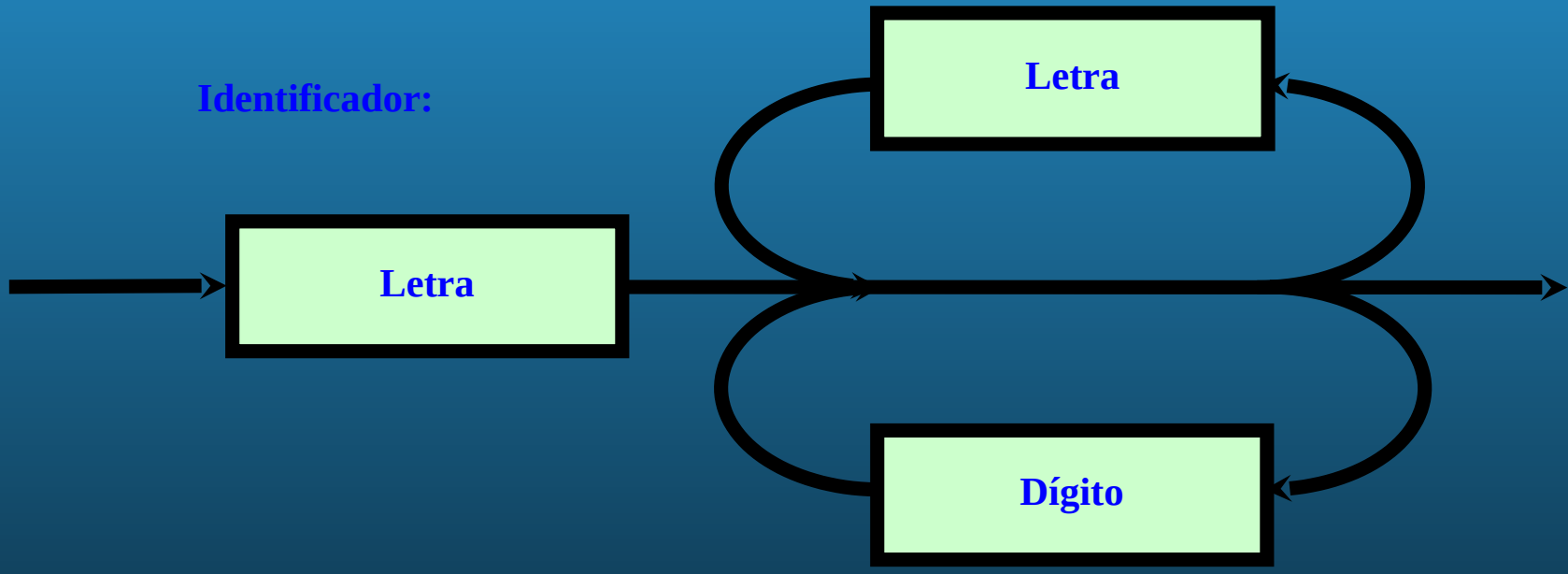
Los ***tipos de valores almacenables*** corresponden a los ***tipos de valores representables***.

Así *variables* ***enteras*** *almacenarán* ***valores enteros***, *variables* ***reales*** *almacenarán valores* ***reales***, etc.

Identificador

Es un texto utilizado para denominar una entidad dentro del **programa** (*procedimiento, variable, constante con nombre, etcétera*).

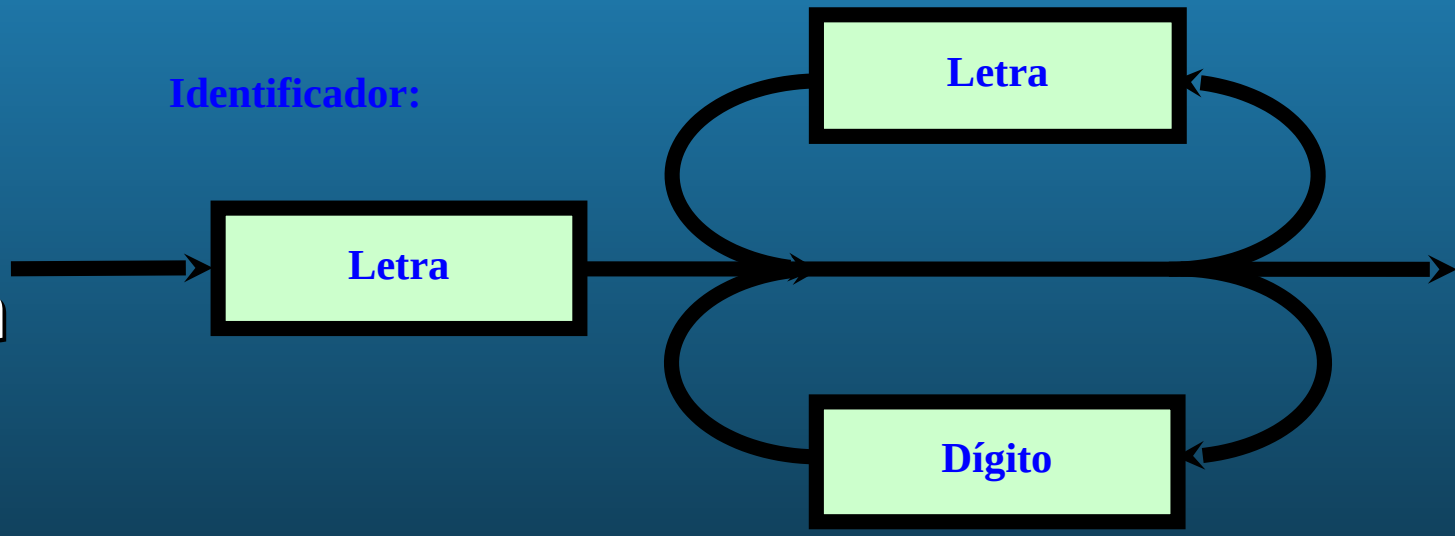
- Está compuesto por **letras** y **dígitos**, pero *siempre comenzando con una **letra***.



Identificador...

Rotula o **etiqueta** un **objeto**, sea éste una variable, un procedimiento, una función y aún al mismo programa principal.

- Los **identificadores deben ser únicos** (*no pueden estar repetidos*) en su **contexto**.
- Ejemplos válidos:
  - xYz
  - C3PO
  - sumatoria



Sobre los identificadores

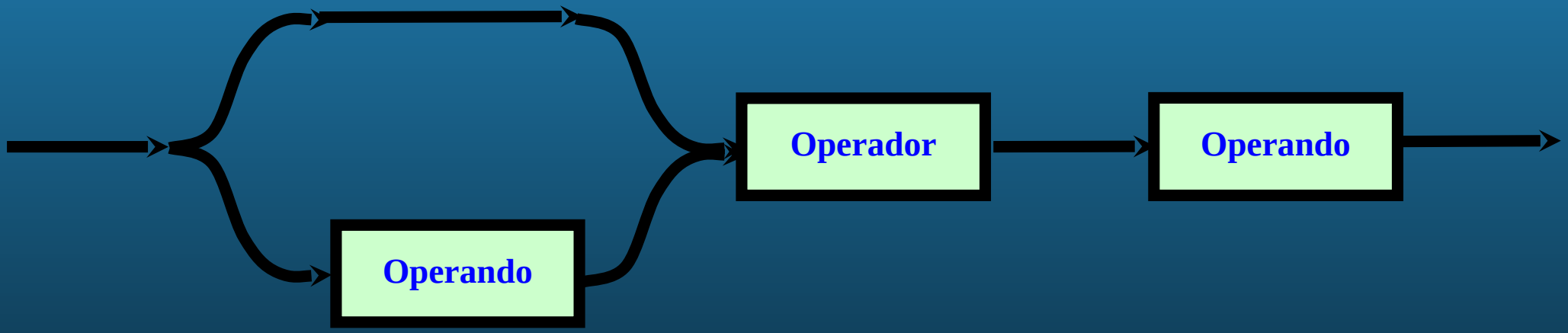
Un identificador **debe** tener en cuenta que ***el usuario del mismo es el propio programador***. Por lo tanto debe vincular el nombre con su **significado** de manera **unívoca** e **inequívoca**.

- Una simple regla: utilizar nombres ***significativos***.
- El tamaño del nombre *depende del* ***contexto***, pero debe ser **suficiente** para que se *asocie unívocamente con su* ***significado***.
- El tiempo insumido ***se compensa más que sobradamente con mayor ausencia de errores***.

Expresiones

Las expresiones son **conjuntos** de **operandos** (*funciones*, *constantes* y/o variables) y **operadores** que **simbolizan una secuencia de** operaciones que **producen un único** resultado final.

Operación:

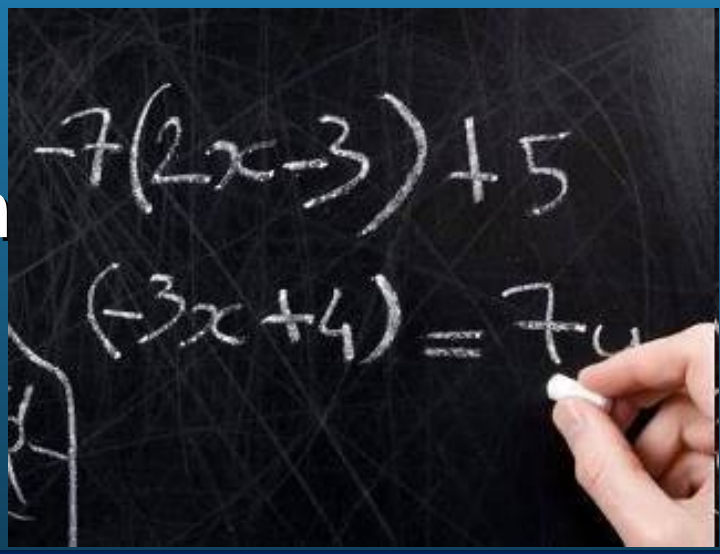


Expresiones...

Cada expresión se resuelve ***de izquierda a derecha***, pero esto puede cambiarse por el uso de **paréntesis** o por la distinta **prioridad** de los diferentes **operadores**.

Los tipos más usuales de expresiones son:

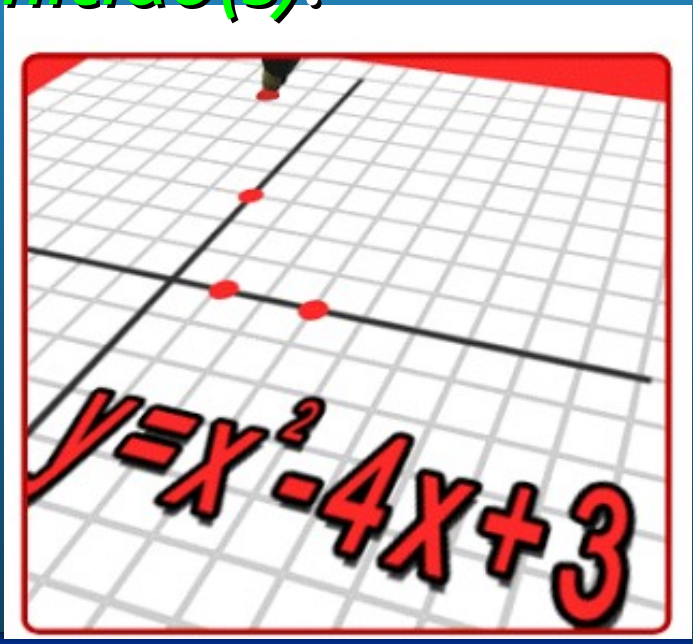
- Alfanuméricas
- Aritméticas o Numéricas
- Relacionales o de Comparación
- Lógicas



Expresiones...

Lo **esencial** de una expresión es:

- Cuáles son sus **operadores**.
  - …y su ***prioridad***.
- Cuáles son sus **operandos**.
  - …y el/los ***tipo(s) de datos admitido(s)***.
- Cuál es el **tipo** de su resultado.



Expresiones Aritméticas

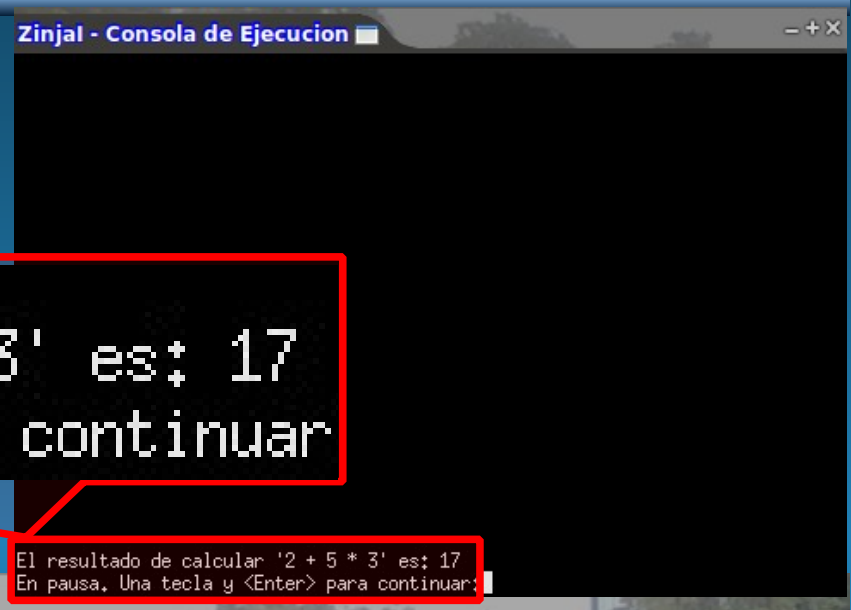
Operador	Prioridad	Operando(s)	Resultado	Ejemplo
**	1	2 números	número	2+5*3 (***17***)
*	2			
/				
+	3			
-				

Expresiones Aritméticas...

Ejemplo

2+5*3 (17)

```
El resultado de calcular '2 + 5 * 3' es: 17
En pausa. Una tecla y <Enter> para continuar
```



Zinjal - 01 - Secuencias

Archivo Editar Ver Ejecucion Depuracion Herramientas Ayuda

Arbol de Archivos

- Fuentes
  - main.cpp
- Cabeceras
  - program1.h
- Otros

Proyecto

main.cpp | program1.h

```
1  // Archivo de traducción de seudocódigo a C++
2  #include "../program1.h"
3
4  principal                                          // Unidad de programa principal
5  //Declaraciones:
6  limpiar;                                           // Limpia la pantalla.
7  //Código:
8  mostrar << "El resultado de calcular '2 + 5 * 3' es: "<< 2 + 5 * 3;
9  pausa;                                             // Pausa antes de finalizar.
10 finPrincipal
11
```

Nota: el operador de concatenación no está definido en todos los lenguajes. Por ejemplo en “C” no existe como tal. En su lugar existen funciones que cumplen dicha tarea.

Operador	Prioridad	Operando(s)	Resultado	Ejemplo
+	1	2 textos	texto	“hola”+”che” (“holache”)

Expresiones Relacionales

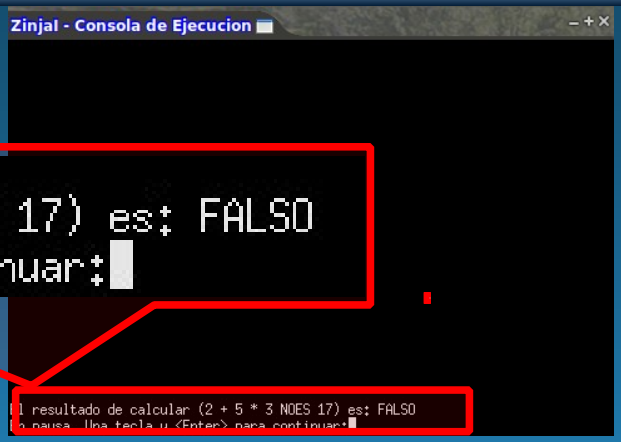
Operador	Prioridad	Operando(s)	Resultado	Ejemplo
<	4	2 números o 2 textos	lógico	2+5*3 NOES 17 (FALSO)
<=				
== ES				
!= NOES				
>=				
>				

Expresiones Relacionales...

Ejemplo

2+5*3 !=17
(FALSO)

```
El resultado de calcular (2 + 5 * 3 NOES 17) es: FALSO
En pausa. Una tecla y <Enter> para continuar:
```



on Depuracion Herramientas Ayuda

main.cpp* program1.h

```
1  // Archivo de traducción de seudocódigo a C++
2  #include "../program1.h"
3
4  principal                                          // Unidad de programa principal
5  //Declaraciones:
6  limpiar;                                          // Limpia la pantalla.
7  //Código:
8  mostrar << "El resultado de calcular (2 + 5 * 3 NOES 17) es: "<< fSi((2 + 5 * 3 NOES 17),"VERDADERO","FALSO");
9  pausa;                                            // Pausa antes de finalizar.
0  finPrincipal
1
```

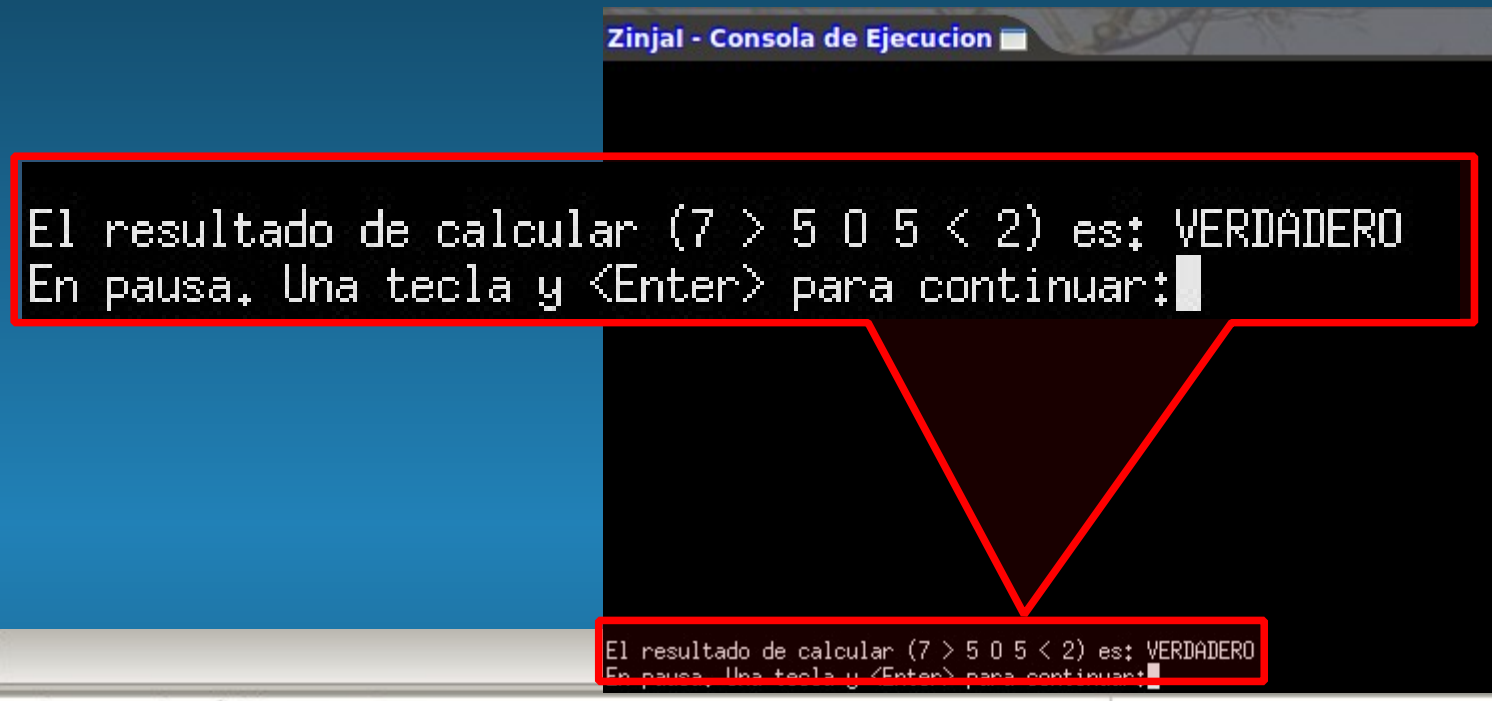
Expresiones Lógicas

Operador	Prioridad	Operando(s)	Resultado	Ejemplo
NO !	5	1 lógico	lógico	NO 17 > 5 (FALSO)
Y &&	6	2 lógicos		17 > 5 Y 5 < 2 (FALSO)
O ||	7			17 > 5 O 5 < 2 (CIERTO)

Expresiones Lógicas...

Ejemplo

17 > 5 O 5 < 2
(CIERTO)



ain.cpp | program1.h

```
// Archivo de traducción de seudocódigo a C++
#include "../program1.h"

principal                                          // Unidad de programa principal
//Declaraciones:
limpiar;                                           // Limpia la pantalla.
//Código:
mostrar << "El resultado de calcular (7 > 5 O 5 < 2) es: "<< fSi((7 > 5 O 5 < 2),"VERDADERO","FALSO");
pausa;                                             // Pausa antes de finalizar.
finPrincipal
```

Validación de Expresiones

Deben efectuarse las siguientes validaciones:

- Expresión bien formada (EBF).
  - Implica un **orden** de **cálculo** ***único***.
- Tipo de datos de cada resultado.
  - Implica **demostrar** que cada **cálculo** es ***viable***.
- Condiciones de validez de cada cálculo.
  - Deben evitarse ***divisiones por cero***, por ejemplo.

Validación de Expresiones (1)

Expresión bien formada (EBF).

- **Implica** un **orden de cálculo** ***único***.
  - Por ejemplo, en la expresión: $b - a / c \uparrow d$
  - El orden de cálculo de los resultados es:
    - R1 ← c ↑ d
    - R2 ← a / R1
    - R3 ← b - R2

Validación de Expresiones (2)

Tipo de datos de cada resultado.

- **Implica** demostrar que cada cálculo *es* ***viable***.
  - Por ejemplo, en la expresión: $b - a / c \uparrow d$
  - Si fueran **a**, **b** y **c** de tipo ***entero*** y **d** fuera ***real***.
    - El tipo de cada resultado es:
      - R1 ← c ^ d *(real)*
      - R2 ← a / R1 *(real)*
      - R3 ← b – R2 *(real)*

Validación de Expresiones (3)

Condiciones de validez de cada cálculo.

- Deben evitarse operaciones no válidas.
  - Por ejemplo, en la expresión: $b - a / c \uparrow d$
  - Si fueran **a**, **b** y **c** de tipo ***entero*** y **d** fuera ***real***.
    - Las condiciones de validez son:
      - R1 ← c ^ d *(real)* <=> c > 0
    - Pues las potencias se calculan:

$$z = x^y \rightarrow$$
$$\ln(z) = y * \ln(x) \rightarrow$$
$$e^{\ln(z)} = z = e^{y * \ln(x)}$$

Validación de Expresiones (3)

Condiciones de validez de cada cálculo.

- Deben evitarse operaciones no válidas.
  - Por ejemplo, en la expresión: $b - a / c \uparrow d$
  - Si fueran **a**, **b** y **c** de tipo ***entero*** y **d** fuera ***real***.
    - Las condiciones de validez son:
      - R1 ← c ^ d <=> c > 0
      - R2 ← a / R1 <=> **R1 ≠ 0**
    - Combinando:
      - **R1 ≠ 0 => c ≠ 0**
    - Condición necesaria: c > 0

Ejercitación

- Resuelva:

$$a - b*c \uparrow d \geq a*b \uparrow (b*c - a/d) \; Y \; b \uparrow 2 - 4*a*c \geq d$$

- Construya una expresión lógica con no menos de 10 operadores y resuélvala, especificando sus condiciones de validez.