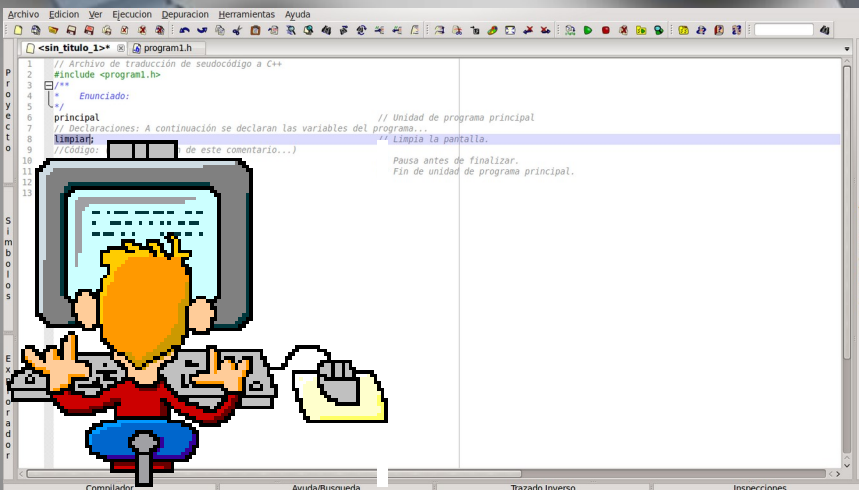




Programación I

Ing. Carlos R. Rodríguez

**Tecnicatura
Universitaria en
Programación**

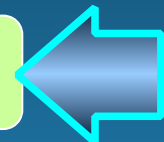
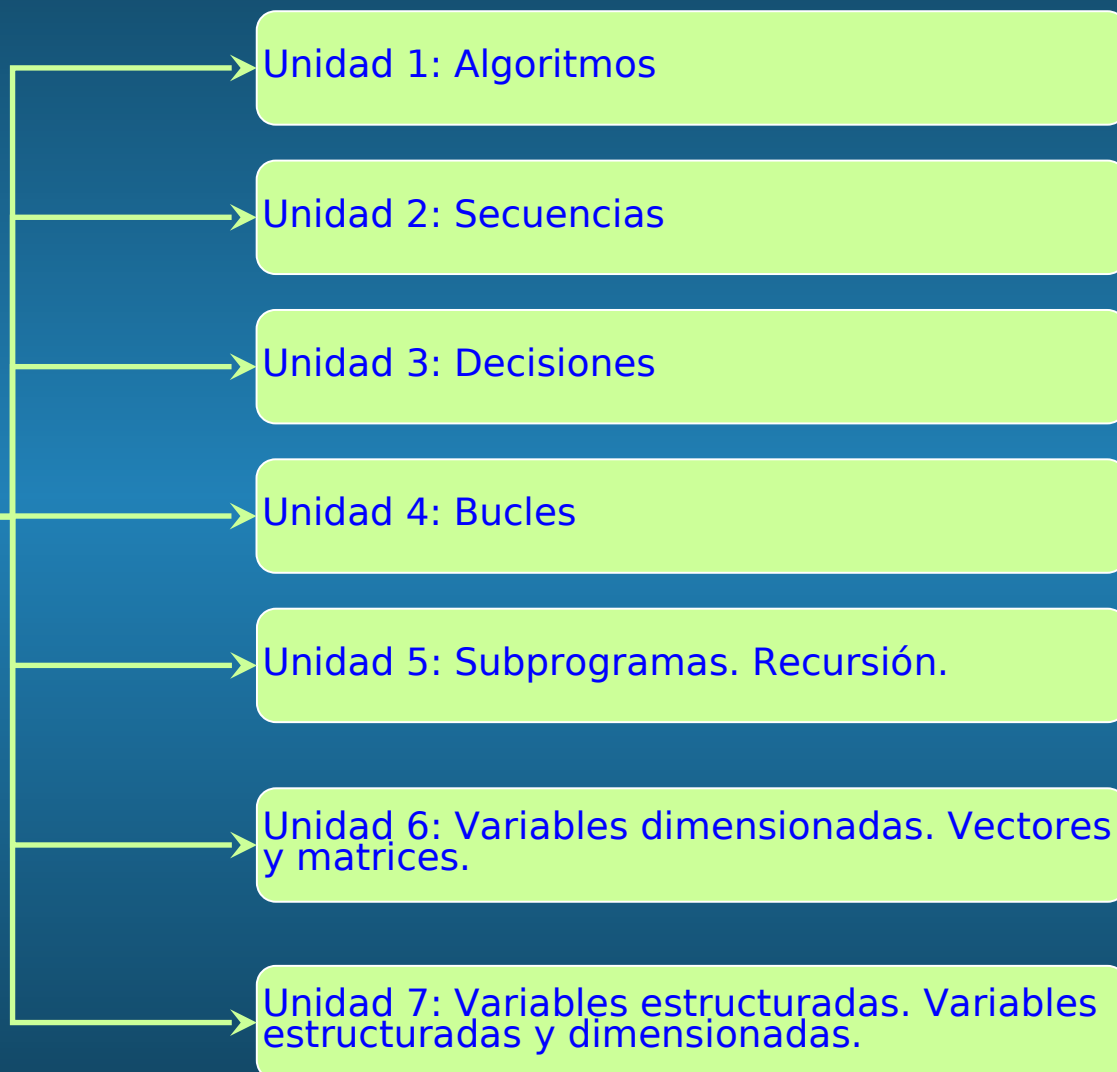


UNIVERSIDAD TECNOLÓGICA NACIONAL
Facultad Regional Mendoza



Programación I

Con pseudocódigo





El tipo de datos de una expresión está dado por la **última operación calculada**.

○ $4 + 15 * 3$

➤ Será *aritmética*

○ $4 + 15 * 3 < 90$

▮ Será *relacional*

○ $4 + 15 * 3 < 90 \text{ Y } 2 * 11 > 17$

▮ Será *lógica*

$$5a^2b^3 + \sqrt{a^2 + b^2}$$

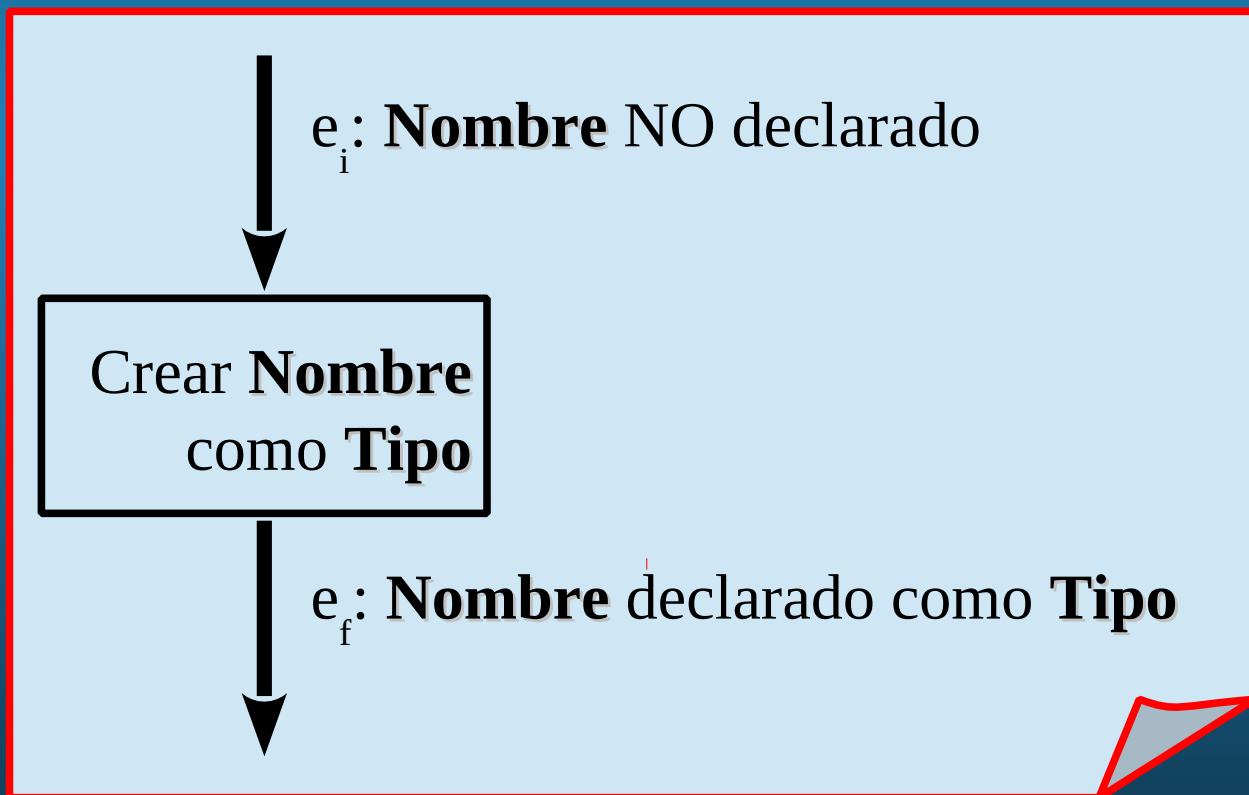
$$\sqrt{4ab^2} + (a + b)^2$$

$$\frac{a^3b}{2ab^2} - \sqrt{12d^4}$$



Para **declarar** un nombre y **asignarle** un tipo de datos:

- La **precondición** será que **no exista una variable con el mismo nombre**, lo que implica que *no se puede redefinir el tipo de una variable*.
- El **estado final** o **poscondición** de la declaración es la existencia **de la variable bajo un nombre no repetido y perteneciendo a un tipo de datos válido**.

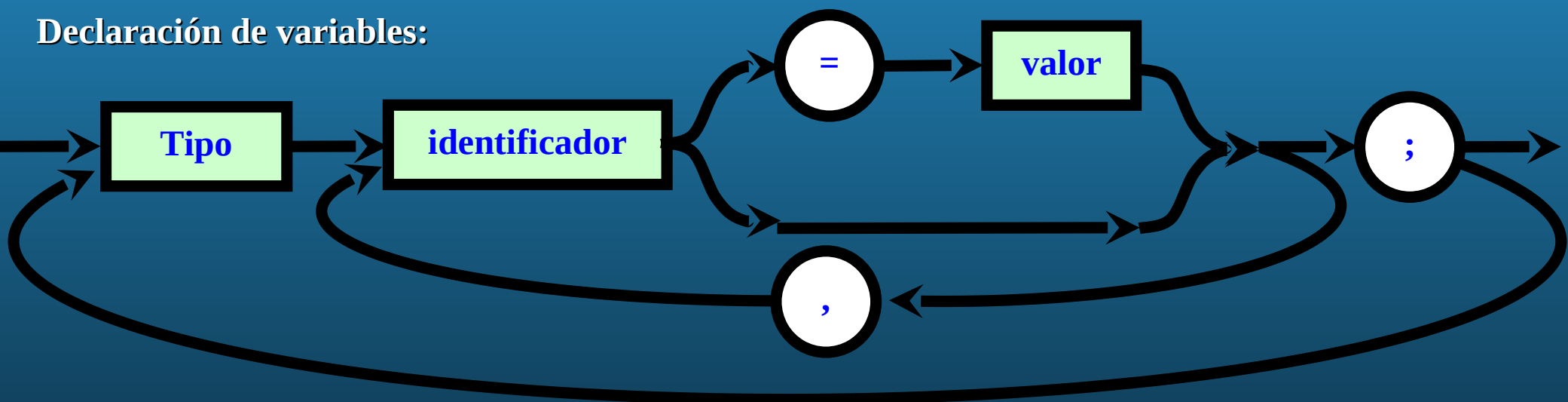




Declarar una variable no implica *necesariamente* definir su **valor inicial**, que entonces permanecerá **indefinido** hasta una **acción explícita** de definición del valor almacenado (por asignación o por lectura).

- Sintaxis:

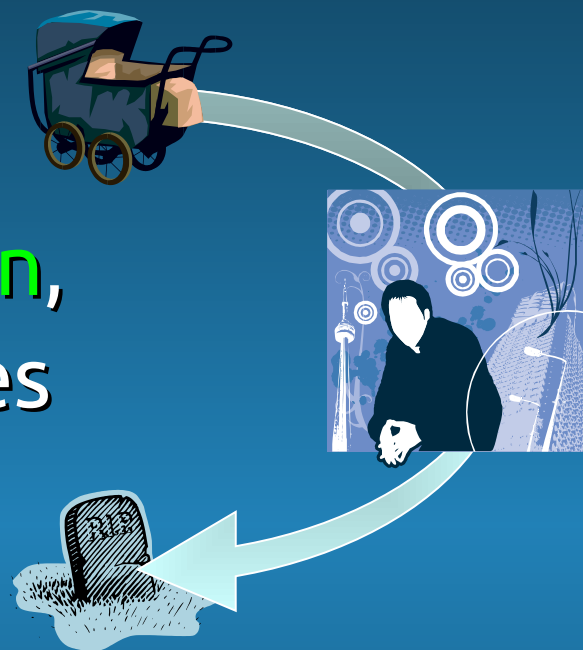
Declaración de variables:





Ciclo de vida:

- Cada **variable**:
 - ▮ Se **crea** a partir de su **declaración**,
 - ▮ **Ocupa** su espacio en **memoria**, es **leída** o **asignada** y...
 - ▮ Finalmente **desaparece**.



Toda variable **existe** mientras **existe** en memoria el **contexto** que le dio origen:

- Programa Principal
- Subprograma
- Bloque {



Visibilidad:

- Es el **ámbito de validez** de su **declaración**:
 - ▮ Dentro de un **procedimiento** o **función** será *local, visible para el código dentro de dicha **unidad de programa**.*
 - ▮ Fuera de toda **unidad de programa** se considerará **global** y será *visible desde cada **unidad de programa** colocada **después de ella**.*





Visibilidad:

- La **primer** variable mostrada es local.
- La **segunda** es global.

2,5 20

En pausa. Una tecla y <Enter> para continuar: |

Zinjal

Archivo Editar Ver Ejecucion Depuracion Herramientas Ayuda

Depuracion: Ejecutando...

main.cpp

```
1 // Archivo de traducción de pseudocódigo a C++
2 #include "../program1.h"
3
4 entero a = 10, b = 20;
5 principal
6 real a = 2.5;
7 limpiar;
8 mostrar << a << tabulado << b << salto;
9 pausa;
10 finPrincipal
```

Zinjal - Consola de Ejecucion

2,5 20

En pausa. Una tecla u <Enter> para continuar: |

```
// Variables globales.
// Unidad de programa principal
// Variable local
// Limpia la pantalla.

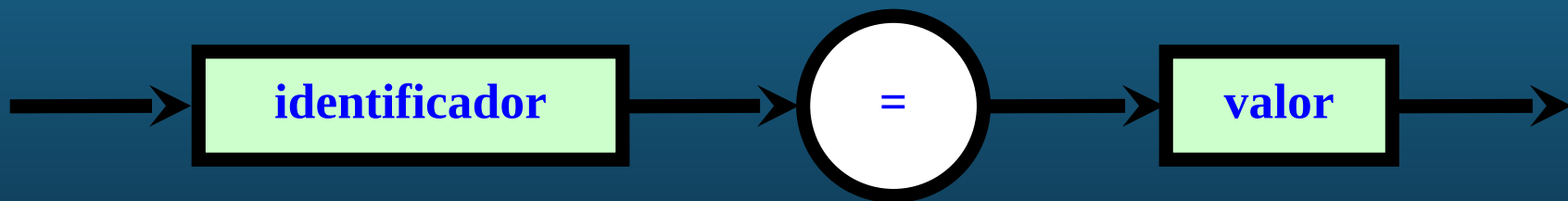
// Pausa antes de finalizar.
// Fin de unidad de programa princ
```




Es la acción por la cual ordenamos **guardar un valor en una variable**.

- Se utiliza el operador "**=**".
- El **valor** ubicado "*a la derecha*" se **carga** (*asigna*) en la **variable** ubicada "*a la izquierda*".
 - ▮ Es una **constante válida** o bien un **cálculo** (*expresión*) que devuelve un **valor válido**.
- Sintaxis:

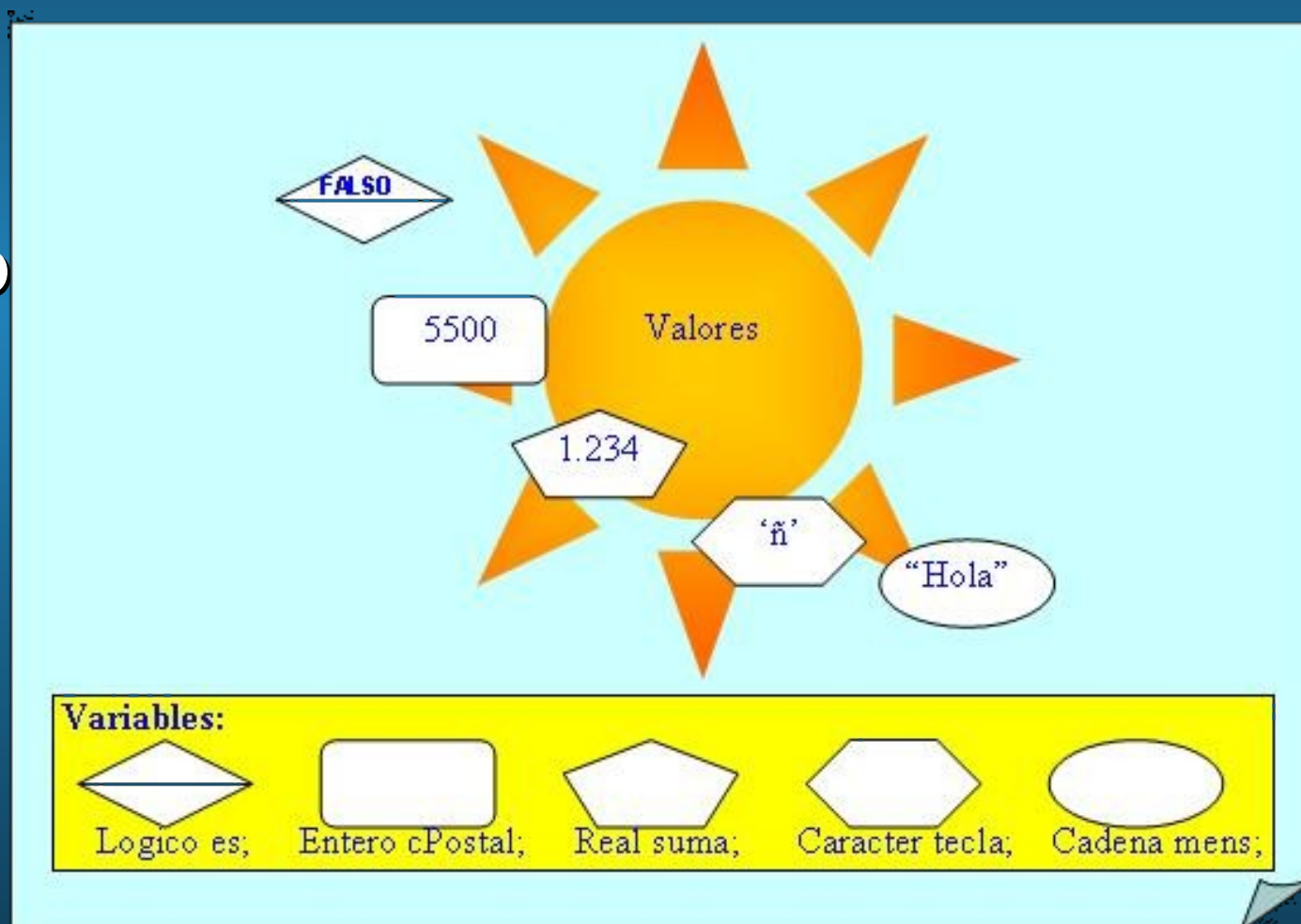
Asignación:





Precondiciones:

- Variable **declarada**
- Valor del **mismo tipo** o **compatible** con el de la variable donde va a almacenarse.





Poscondición:

- $\text{Variable} \equiv \text{valor} \Leftrightarrow \text{tipo}(\text{Variable}) \equiv \text{tipo}(\text{valor})$
 - ▮ El **valor almacenado** es igual al calculado, siempre y cuando *ambos tipos de datos sean idénticos*.
- $\text{Variable} \approx \text{valor} \Leftrightarrow \text{tipo}(\text{Variable}) \approx \text{tipo}(\text{valor})$
 - ▮ Los **valores** son **similares** en el caso que los tipos de datos *sean compatibles* pero *no iguales*.
- $\text{Error} \Leftrightarrow \text{tipo}(\text{Variable}) \neq \text{tipo}(\text{valor})$





Un programa debe comunicarse con el mundo exterior a él:

- Los dispositivos que utiliza para ello se denominan **periféricos**:

- ▮ Por ejemplo: el **teclado**.
- Los datos fluyen desde el exterior hacia la memoria.
 - ▮ Implica almacenar los **valores** ingresados en variables.





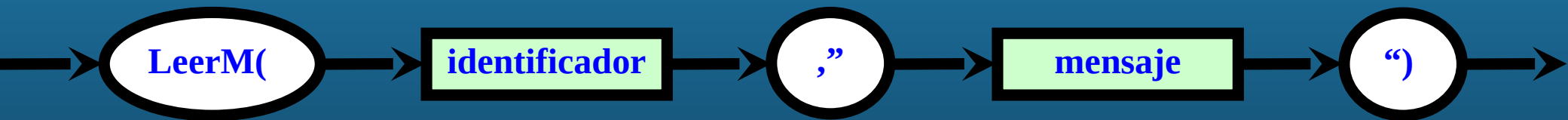
Sintaxis:



Lectura:

Puede ser conveniente utilizar su **variante con mensaje**:

Lectura con mensaje:



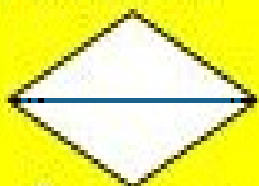


Precondiciones:

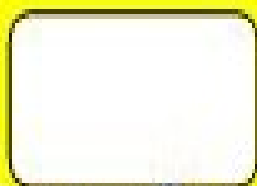
- **Variable declarada.**

- ▮ Debe existir el **nombre** y tener **asignado** un **tipo de datos**.

Variables:



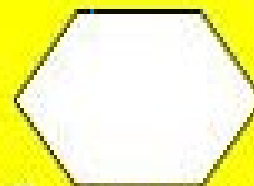
Logico es;



Entero cPostal;



Real suma;



Caracter tecla;



Cadena mens;



Poscondición:

- $\text{Variable} \equiv \text{valor} \Leftrightarrow \text{tipo}(\text{Variable}) \equiv \text{tipo}(v\text{Leído})$
 - ▮ El **valor almacenado** es igual al leído, siempre y cuando *ambos tipos de datos sean idénticos*.
- $\text{Variable} \approx \text{valor} \Leftrightarrow \text{tipo}(\text{Variable}) \approx \text{tipo}(v\text{Leído})$
 - ▮ Los **valores** son **similares** en el caso que los tipos de datos *sean compatibles pero no iguales*.
- $\text{Error} \Leftrightarrow \text{tipo}(\text{Variable}) \neq \text{tipo}(v\text{Leído})$





En este caso los **datos** fluyen desde la **memoria** hacia el **exterior**.

- Para esta **acción** es necesaria una **lista no vacía de valores** (*posiblemente expresados bajo la forma de una **lista de expresiones***), que son enviados al **periférico** (*asumiremos **pantalla***).

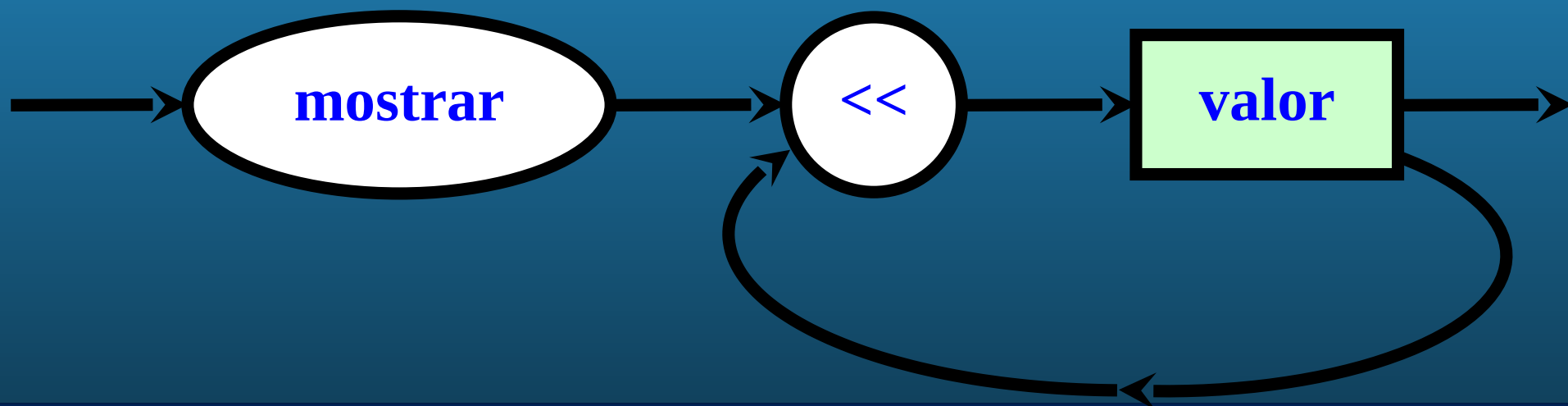




Sintaxis:

- Puede considerarse que **cada valor se escribe por separado**, pudiendo abreviar escrituras sucesivas bajo la forma de una lista cuyo separador es "<<".

Escritura:





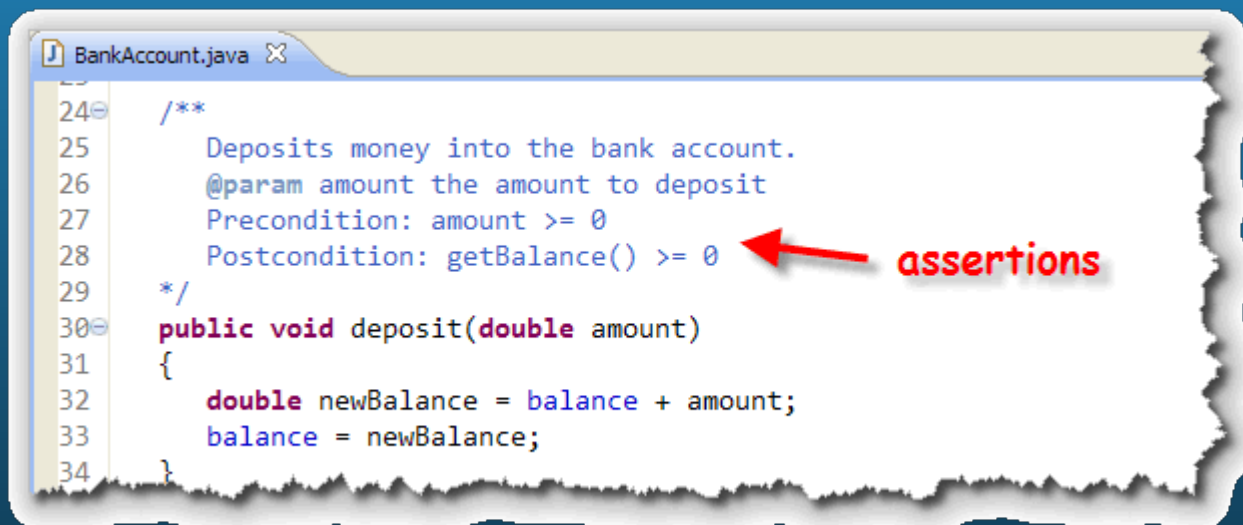
Precondición:

- Cada expresión debe ser **válida** (*calculable*).

Poscondición:

- El estado final consiste en la lista de valores *incluida en el conjunto de resultados* (*salidas*).

Esta será - en principio - **la única forma** que tienen los programas de **producir resultados** (*escribirlos*).



```
BankAccount.java X
24  /**
25     Deposits money into the bank account.
26     @param amount the amount to deposit
27     Precondition: amount >= 0
28     Postcondition: getBalance() >= 0
29  */
30  public void deposit(double amount)
31  {
32      double newBalance = balance + amount;
33      balance = newBalance;
34  }
```

← assertions



Como el **objetivo** de cualquier **programa** real es *producir resultados*, tres cosas son evidentes:

- Sin al menos **una instrucción de escritura**, *no es un algoritmo*.
- La **verificación del cumplimiento de especificaciones** de un programa **puede hacerse partiendo de sus acciones de escritura**.
- El código ubicado después de la última escritura **es siempre código muerto**, *sin efecto alguno*.



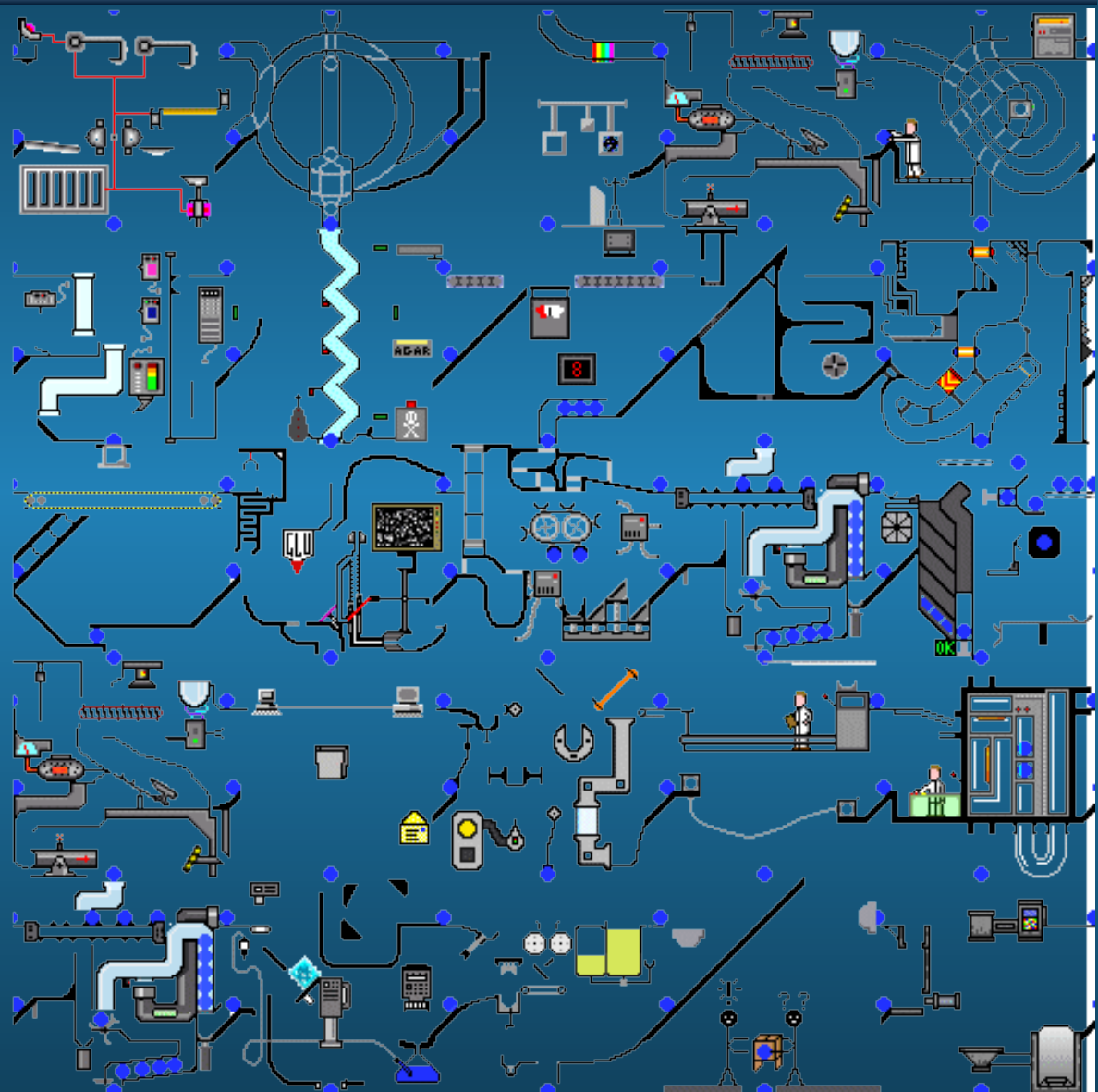


Un procedimiento se define como **aquella parte de un programa** (*un subprograma*) **que realiza una acción separada y perfectamente identificable**, tal como entrada de datos, cálculo, etc.





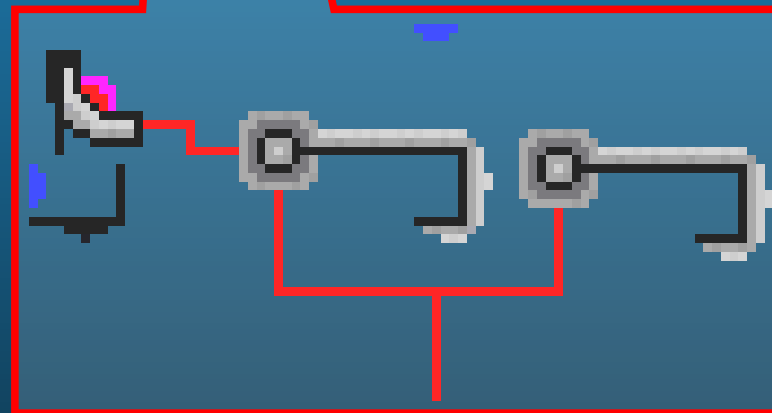
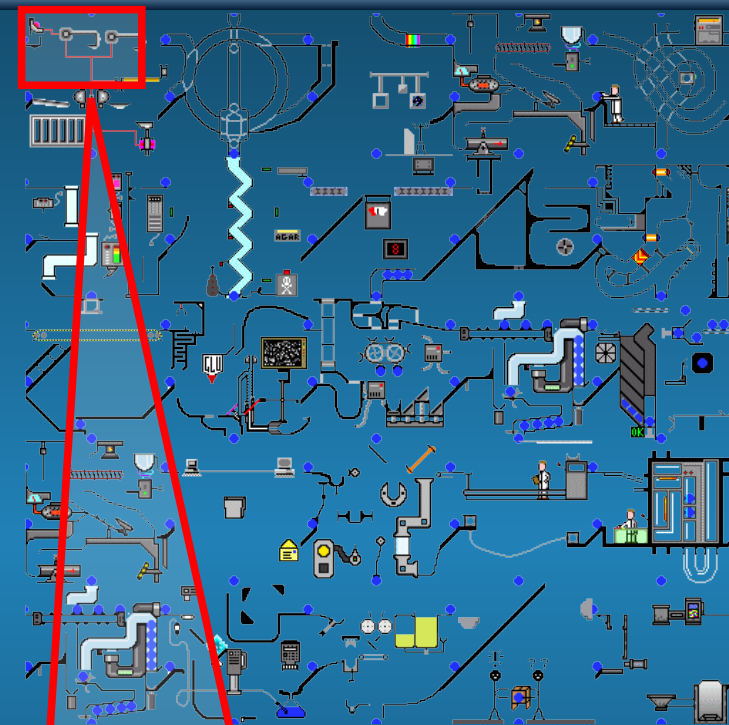
A su vez, cada procedimiento *puede llamar a otros* subprogramas, de forma similar a como éste *fue llamado desde el programa principal o desde un procedimiento de nivel superior.*





El mecanismo de **dividir un problema dado en subproblemas cada vez más pequeños, que a su vez van de lo general a lo particular** es conocido como **diseño descendente**.

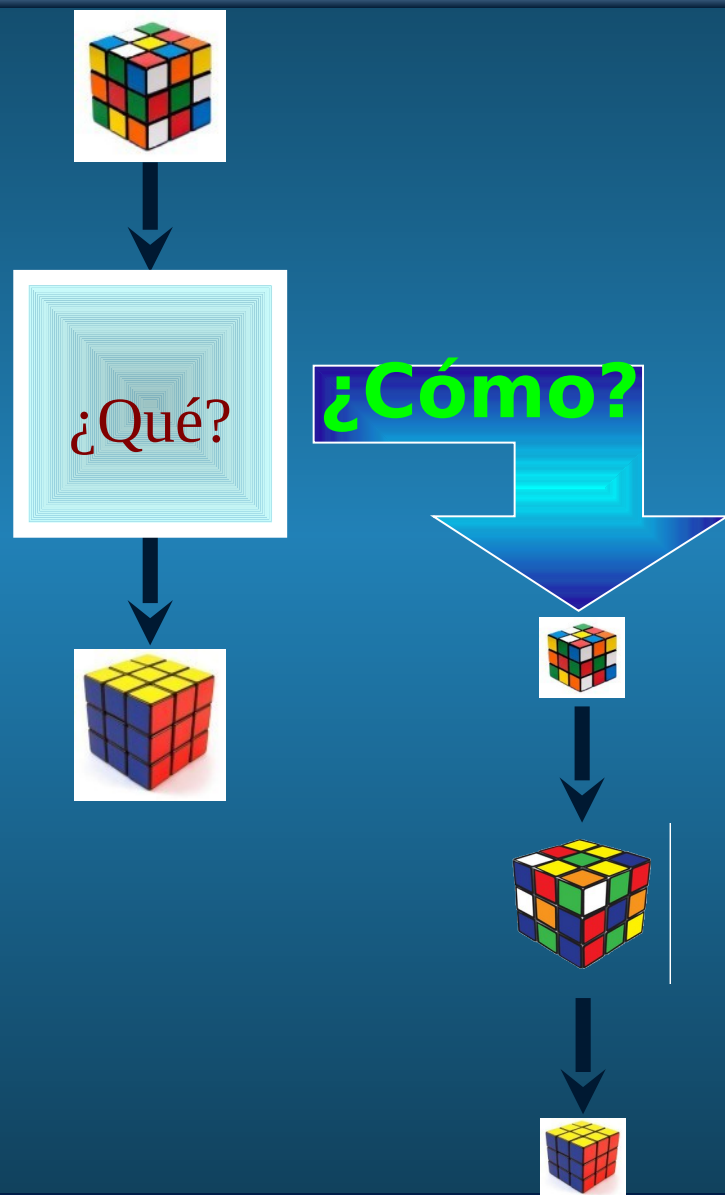
- Permite **mantener cada subprograma dentro de un tamaño manejable** por el programador.





La llamada a un procedimiento es una **instrucción secuencial** vista desde el punto de vista de la **unidad de programa que efectúa la llamada**.

- Es decir, *haciendo abstracción de la secuencia de instrucciones simples y/o compuestas que implica la ejecución del procedimiento nombrado por la llamada*.





Veamos un ejemplo:

- Pretendemos intercambiar 2 valores numéricos.
 - ▮ Sabemos que **debe hacerse**, pero en este instante **desconocemos cómo podría lograrse**.

```
1 // Archivo de traducción de pseudocódigo a C++
2 #include "../program1.h"
3 /*
4  * Enunciado: Dados dos valores numéricos, intercambiarlos.
5  */
6 procedimiento intercambiar(real porRef, real porRef); // Prototipo de llamada (dos argumentos de punto flotante por referencia)
7
8 principal // Unidad de programa principal
9 real uno,dos; // Datos (variables locales)
10 limpiar; // Limpia la pantalla.
11 leerM(uno,"Primer valor:");
12 leerM(dos,"Segundo valor:");
13 intercambiar(uno,dos); // Envía dos argumentos reales por referencia para: uno <-> dos
14 mostrar << "Primer valor:" << uno << tabulado << "Segundo valor:" << dos << salto;
15 pausa; // Pausa para mostrar los resultados.
16 finPrincipal // Fin de unidad de programa principal.
```




Veamos un ejemplo:

- Pretendemos intercambiar 2 valores numéricos.
 - ▮ Luego debe especificarse el código que *resuelve (como) la llamada (que)*.

```
procedimiento intercambiar(real porRef a, real porRef b) {           // Recibe 2 parámetros por referencia
    real c = a;
    a = b;
    b = c;
}
```

```
3  /*
4  Enunciado: Dados dos valores numéricos, intercambiarlos.
5  */
6  procedimiento intercambiar(real porRef, real porRef);           // Prototipo de llamada (dos argumentos de punto flotante por referencia)
7
8  principal                                                       // Unidad de programa principal
9  real uno,dos;                                                    // Datos (variables locales)
10 limpiar;                                                         // Limpia la pantalla.
11 leerM(uno,"Primer valor:");
12 leerM(dos,"Segundo valor:");
13 intercambiar(uno,dos);                                           // Envía dos argumentos reales por referencia para: uno <-> dos
14 mostrar << "Primer valor: " << uno << tabulado << "Segundo valor: " << dos << salto,
15 pausa;                                                           // Pausa para mostrar los resultados.
16 finPrincipal                                                     // Fin de unidad de programa principal.
```



Veamos un ejemplo:

- Pretendemos intercambiar 2 valores numéricos.
 - ▮ Ya se puede ejecutar.

Zinjal - Consola de Ejecucion

```
Valor 1:1234.56  
Valor 2:65.4321  
Valor 1:65.4321 Valor 2:1234.56  
En pausa. <Escape> para continuar...
```




Resolver:

- Teniendo como **datos** la **base** y la **altura** de un triángulo, calcular su superficie.

▮ Sabiendo que:

$$\text{Superficie} = \frac{\text{base} * \text{altura}}{2}$$



- ▮ Determine las **condiciones de validez** del programa.