# TEMA 1: Léxico y organización de un algoritmo

**1.1** Léxico de un algoritmo

**1.2** Tipos de datos primitivos: dominio y operaciones

**1.3** Acciones primitivas

**1.4** Organización de las acciones

**1.5** Tipos de datos no primitivos

# 1.1 Léxico de un algoritmo

- **Léxico de un algoritmo: INFORMACIONES + ACCIONES**
- **INFORMACIONES ≡ VARIABLES**
- **Magnitudes que caracterizan un proceso algorítmico**
- **CONSTRUIR UN ALGORITMO** CONSISTE EN ELEGIR UN CONJUNTO DE **INFORMACIONES** Y OTRO DE **ACCIONES,** Y A CONTINUACIÓN DECIDIR EL MODO DE ORGANIZAR LAS ACCIONES EN EL TIEMPO PARA OBTENER EL RESULTADO DESEADO POR ACUMULACIÓN DE SUS EFECTOS

# 1.1 Léxico de un algoritmo

- **COMPRAR UNA ENTRADA PARA IR A LOS TOROS:**

**Informaciones**: la entrada, tipo de entrada (sol, sombra, tendido...), disponibilidad de la entrada

**Acciones:** ir a la taquilla, elegir la entrada, preguntar si quedan, si quedan comprar, si no se puede elegir otra, repetir hasta que se consiga una entrada o desistimos de ir a los toros.

- Necesitamos una notación: **Notación algorítmica**

# 1.1 Léxico de un algoritmo

**LA NOTACIÓN ALGORÍTMICA FIJA LA FORMA DE:**

- Describir las acciones
- Describir las informaciones
- Organizar las acciones en el tiempo
- Incluye acciones elementales

**ALGORITMO:**

- LÉXICO: informaciones u objetos y acciones
- CONTROL: ordenar en el tiempo cómo actúan las acciones sobre los objetos

**Abstracción:**

- Mecanismo fundamental para dominar la complejidad cuando programamos. “Eliminar detalles innecesarios y considerar lo esencial”. El léxico fija el nivel de abstracción

# 1.1 Léxico de un algoritmo

**Construir algoritmos:**

- Fijar el léxico
- Organizar las acciones en el tiempo mediante: **SECUENCIACIÓN, ANÁLISIS DE CASOS E ITERACIÓN (RECURSIÓN)**

**LÉXICO**

**Dada la especificación de un problema hay que:**

- Elegir y nombrar las informaciones
- Asociar un tipo a cada información
- Elegir y nombrar las acciones
- Asociar una precondición y una postcondición a cada acción.

# 1.1 Léxico de un algoritmo

- información (o variable) : nombre y tipo
- acción: nombre, precondición y postcondición

- **TIPO DE DATO:** dominio de valores y acciones que son posibles realizar sobre esos valores.
- **PRECONDICIÓN**: requerimiento de la acción
- **POSTCONDICIÓN**: efecto de la acción

# Estructura de un algoritmo

```
LÉXICO
        // Declaraciones de tipos, variables, constantes y acciones
        v1: tipo
        A1: una acción
                PRE { precondición A1 }
                POST { postcondición A1 }
        LÉXICO
                // Declaraciones de tipos, variables, constantes y
                // acciones
        ALGORITMO
                // Secuencia de instrucciones
        FIN
ALGORITMO
        PRE { precondición algoritmo }
        POST { postcondición algoritmo }

        // Secuencia de instrucciones
FIN
```

# 1.2 Tipos de Datos primitivos

- **TIPOS DE DATOS**

  Un tipo de datos especifica un **DOMINIO** de valores y el conjunto de **OPERACIONES** que son aplicables a ese dominio.

- **NUESTRA NOTACIÓN INCLUYE LOS TIPOS DE DATOS:**

  Entero, Real, Booleano, Carácter, Intervalos de enteros, reales y carácter. Así como mecanismos para definir nuevos tipos de datos.

- **Ejemplos:**

  total : Entero;
  i, j : [1,100];
  letra : Carácter;
  esúltimo : Booleano;

# 1.2 Tipos de Datos primitivos

- **Para cada tipo es preciso conocer:**
  - Dominio de los valores
  - Operaciones definidas
  - Sintaxis de los literales
  - Sintaxis de las expresiones

- **Enteros:** cualquier valor entero positivo o negativo válido.

- **Reales:** cualquier valor numérico real positivo o negativo válido. Utilizaremos el símbolo '.' (punto) para separar la parte entera de la parte decimal.

- **Booleanos**: los dos valores lógicos, Verdadero y Falso.

- **Caracteres:** el dominio de este tipo está formado por los caracteres de un código válido y un literal se denota como un carácter encerrado entre apóstrofos.

# 1.2 Tipos de Datos primitivos

| Tipo de dato | Ejemplos de literales |
|---|---|
| Entero | 0, 352, –342, 20050 |
| Real | 4.22, –23.44, 341.015 |
| Booleano | Falso, Verdadero |
| Carácter | ‘A’, ‘a’, ‘$’, ‘1’, ‘+’ |

# 1.2 Tipos de Datos primitivos

| Tipo de dato | Operaciones | |
|---|---|---|
| Entero | - | (Entero → Entero) |
| | +, -, *, DIV, MOD | (Entero x Entero → Entero) |
| | / | (Entero x Entero → Real) |
| | <, >, =, ≤, ≥ , ≠ | (Entero x Entero → Booleano) |
| | Predecesor, Sucesor | (Entero → Entero) |
| Real | - | (Real → Real) |
| | +, -, *, / | (Real x Real → Real) |
| | <, >, = , ≤, ≥, ≠ | (Real x Real → Booleano) |

# 1.2 Tipos de Datos primitivos

| Tipo de dato | | Operaciones |
|---|---|---|
| Booleano | Y, O | (Booleano x Booleano→ Booleano) |
| | NO | (Booleano → Booleano ) |
| Carácter | | |
| | Car | (Entero → Carácter) |
| | Ord | (Carácter → Entero) |
| | <, >, = , ≤, ≥, ≠ | (Carácter x Carácter → Booleano) |
| | Predecesor, Sucesor | (Carácter → Carácter) |

# 1.2 Tipos de Datos primitivos

- **Ejemplo de declaración de variables**

**Léxico**

```
númeroAlumnos : Entero;
cursos : Entero;
media : Real;
nota : Real;
peso : Real;
letra : Carácter;
aprobado : Booleano;
```

- **Posibles valores:**

```
númeroAlumnos = 200,  cursos = 4,  media = 5.0,

nota = 6.0, peso =  80.0, letra = 'A', aprobado = Verdadero
```

# 1.2 Tipos de Datos primitivos

| Expresión | Tipo | Tipo R. | Valor R. |
|---|---|---|---|
| númeroAlumnos DIV cursos | Aritmética | Entero | 50 |
| númeroAlumnos MOD cursos | Aritmética | Entero | 0 |
| 1000 – númeroAlumnos * 2 | Aritmética | Entero | 600 |
| nota * peso / 100.0 – 3.0 | Aritmética | Real | 1.8 |
| nota* (peso/100.0 – 3.0) | Aritmética | Real | -13.2 |
| nota > 7.0 | Relacional | Booleano | Falso |
| (númeroAlumnos DIV cursos) > 20 | Relacional | Booleano | Verdadero |
| letra = ‘B’ | Relacional | Booleano | Falso |
| (nota > 7.0) Y (media = 5.0) | Booleana | Booleano | Falso |
| NO (letra = ’B’) O aprobado | Booleana | Booleano | Verdadero |
| Carácter (66) | Carácter | Carácter | 'B' |
| Sucesor (letra) | Carácter | Carácter | 'B' |

# 1.3 Acciones primitivas

- **LA ACCIÓN DE LA ASIGNACIÓN:**

  La asignación es la acción primitiva que caracteriza a los lenguajes imperativos.

- **Sintaxis**

  **<**nombre de la variable**>** ← **<**expresión**>**

- **Semántica**

  Acción elemental de asignar a la variable cuyo nombre aparece a la izquierda del símbolo ← el resultado de evaluar la expresión de la derecha.

  La acción:

  númeroAlumnos ← 200;

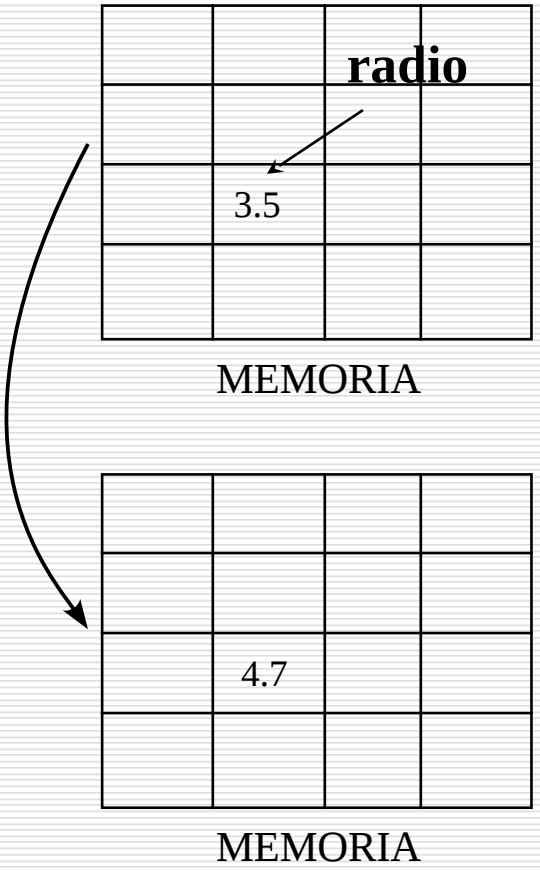
  Asigna a la variable númeroAlumnos el valor 200.

# 1.3 Acciones primitivas

**Ejemplos**

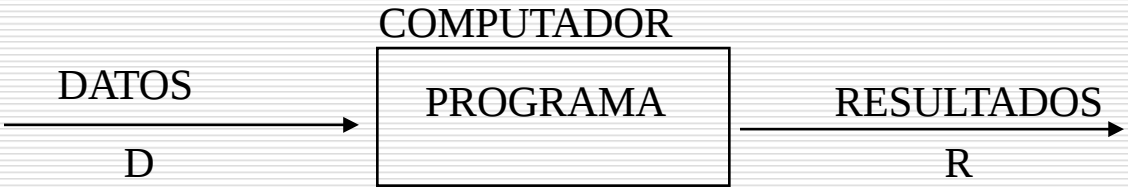
radio ← 3.5;
radio ← radio +1.2;
descuento ← sueldobruto * irpf;
cond1← (p>0) **Y** (cond2=falso);
longitud ← longitud / 2;

- El nuevo valor de longitud es función del anterior.



# 1.3 Acciones primitivas

- **ACCIONES PRIMITIVAS PARA ENTRADA/SALIDA DE DATOS**




- **ENTRADA DE DATOS:** Recibir datos desde un terminal (teclado)

  **Leer (**lista de variables**)**

- **Leer** (radio);

  **Leer** (númeroAlumnos)

  Asignan a la variable entre paréntesis el valor introducido por el teclado

# 1.3 Acciones primitivas

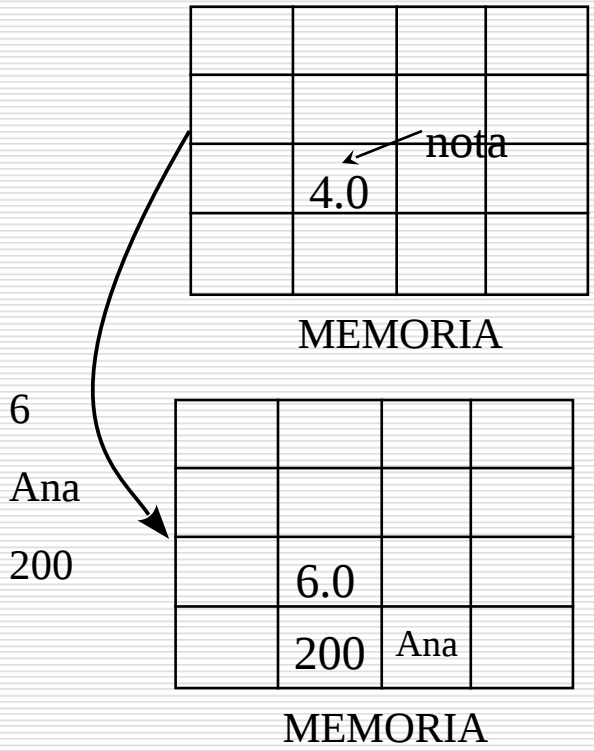
**Ejemplo**

**LÉXICO**

nota: Real;
nombre: Secuencia de caracteres;
numexam: Entero;

**ALGORITMO**

.....
**LEER(**nota, nombre, numexam**)**;
.....



# 1.3 Acciones primitivas

- **SALIDA DE DATOS**: enviar al terminal de salida los valores obtenidos como resultado de evaluar una lista de expresiones

**Escribir** (lista de expresiones)

**LÉXICO**

nota: Real;

nombre: Secuencia de caracteres;

numexam: Entero;

**ALGORITMO**

.....

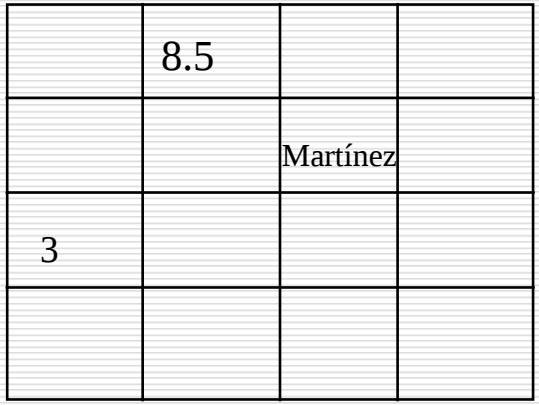
**Escribir (**nota, nombre, numexam**)**;

.....

# 1.3 Acciones primitivas

Supuestas las asociaciones

| **variable** | **valor** |
|---|---|
| nota | 8.5 |
| nombre | Martínez |
| numexam | 3 |




MEMORIA

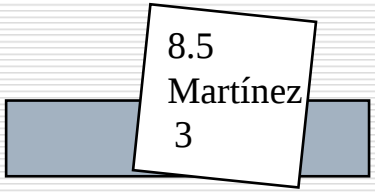
el efecto de

**Escribir** (nota, nombre, numexam)

PANTALLA


es: 8.5 Martínez 3

IMPRESORA



# 1.4 Organización de las acciones

**Composición secuencial:**

Introducción de estados intermedios para reducir la complejidad.

Para ello descomponemos el problema inicial en subproblemas más simples que se pueden resolver de forma independiente

| | | |
|---|---|---|
| $\{E_0\}$ | $\{E_0\}$ | $\{E_0\}$ |
| | | **$A_1$** |
| | **$A_1$** | $\{E_1\}$ |
| **A** | $\{E_1\}$ | **$A_{21}$** |
| | **$A_2$** | $\{E_2\}$ |
| | | **$A_{22}$** |
| $\{E_n\}$ | $\{E_n\}$ | $\{E_n\}$ |

# 1.4 Organización de las acciones: composición secuencial

| | | |
|---|---|---|
| $\{E_0\}$ | $\{E_0\}$ | $\{E_0\}$ |
| | | **$A_1$** |
| | **$A_1$** | $\{E_1\}$ |
| **A** | $\{E_1\}$ | **$A_{21}$** |
| | **$A_2$** | $\{E_2\}$ |
| | | **$A_{22}$** |
| $\{E_n\}$ | $\{E_n\}$ | $\{E_n\}$ |

- El estado inicial ($E_0$) debe cumplir la Precondición y en el estado final ($E_n$) se debe cumplir la postcondición.
- El subproblema $A_1$ debe cumplir la precondición inicial, y la postcondición de este subproblema será la precondición para $A_2$
- Al final la postcondición del último subproblema debe cumplir la postcondición del problema inicial.

# 1.4 Organización de las acciones: composición secuencial

- **Problema:** Escribir un algoritmo que dado un número de segundos inferior a $10^6$, obtenga una magnitud equivalente en días, horas, minutos y segundos.
  Ej. dato: 309639
  resultado: 3, 14, 0, 39

- **Diseño:**
  a) n =86400d+3600h+60m+s
  b) n = ((24d+h)60+m)60+s

  ¿Cómo obtener *d*, *h*, *m* y *s* a partir de esas ecuaciones?

# 1.4 Organización de las acciones: composición secuencial

**LÉXICO**

n: Entero[0,999999]

d: Entero>=0

h: Entero [0,23]

m,s: Entero [0,59]

**ALGORITMO**

Leer(n)

***A -----→ Descomponer***

Escribir(d,h,m,s)

**Descomposición Secuencial:**

**$A_1$** d ← n **div** 86400; rd ← n **mod** 86400;

*{ n= 86400d+ rd, 0≤ rd<86400}*

**$A_2$** h ← rd **div** 3600; rh ← rd **mod** 3600;

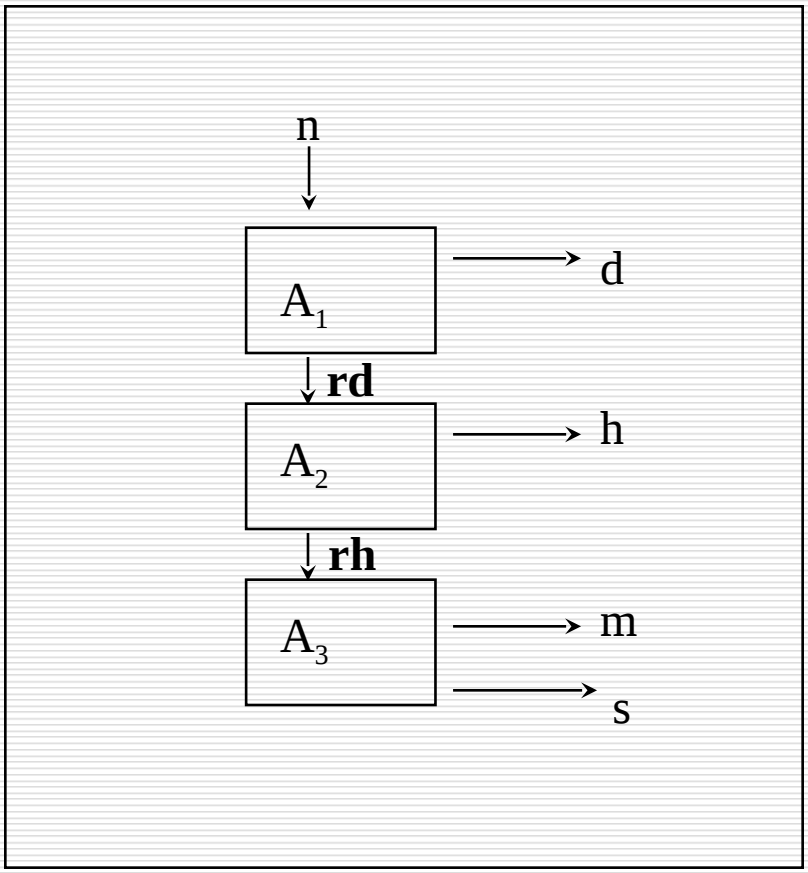
*{ rd= 3600h+ rh, 0≤ rh<3600}*

**$A_3$** m ← rh **div** 60; s ← rh **mod** 60;

*{ n= 86400d+ 3600h + 60 m + s, 0≤d,*

*0 ≤ h< 3600, 0 ≤ m,s < 60}*

# 1.4 Organización de las acciones: composición secuencial




**ALGORITMO**

Leer (n)

**CONVERTIR**

Escribir (d,h,m,s)

*La descomponemos en acciones elementales*

# 1.4 Organización de las acciones: composición secuencial

**LÉXICO**

```
  n: Entero [0,999999]        {dato: nº segundos}
  h: Entero [0,23]            { número de horas}
  m, s :Entero [0,59]         { número de minutos y de segundos}
  d: Entero >=0               { número de días }
  rd: Entero [0,86399]        { resto días }
  rh: Entero [0,3599]         { resto horas }
```

**ALGORITMO**

```
        Leer (n)
        d ← n div 86400; rd ← n mod 86400;
          { n= 86400d+rd,0≤rd<86400}
        h ← rd div 3600; rh ← rd mod 3600;
          { rd=3600h+rh, 0≤rh<3600}
        m ← rh div 60;  s ← rh mod 60;
          { n=86400d + 3600h +60m +s, 0≤h<24, 0≤m,s<60 }
        Escribir (d, h, m, s);
```

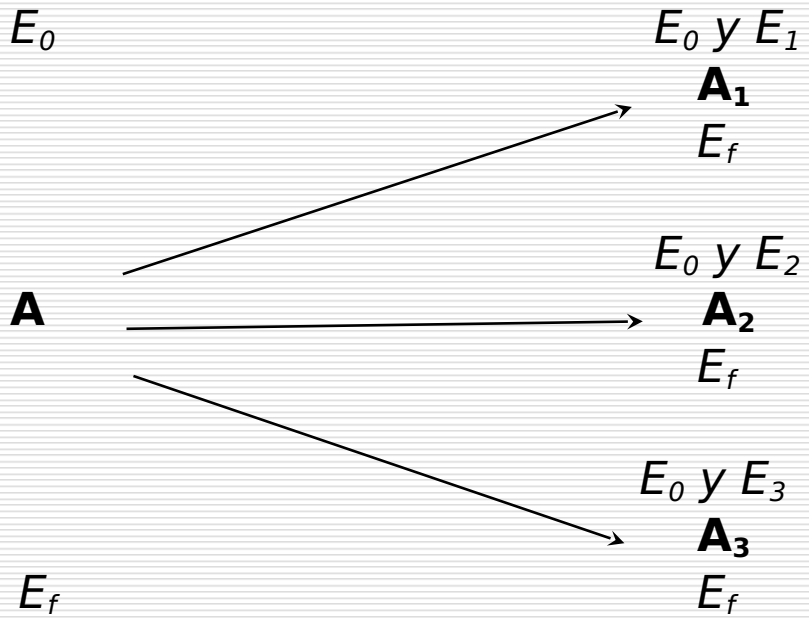
**FIN.**

# 1.4 Organización de las acciones: análisis de casos

- Técnica de descomposición
- Se basa en la partición del dominio de datos en subdominios (casos). Cada subproblema es la restricción del problema inicial al del subdominio considerado
- La descomposición puede estar guiada por la estructura de los datos o de los resultados.

# 1.4 Organización de las acciones: análisis de casos

- La postcondición de cada subproblema debe cumplir la postcondición del problema inicial y la unión de las precondiciones de los subproblemas debe cubrir la precondición del problema inicial.



# 1.4 Organización de las acciones: análisis de casos

- **Enunciado:** Dados dos número enteros calcular el mayor

- **Especificación:**

  x, y, z: entero;

  Precondición: {x = X ^ y = Y}

  Postcondición: { z = max (X, Y) }

- **Lectura de la especificación:**

  Dados tres enteros x, y, z, tal que x contiene un valor **X**, e y un valor **Y**, después de la acción máximo obtenemos en z el máximo de los valores **X** e **Y**.

# 1.4 Organización de las acciones: análisis de casos

- **Análisis:**

  Existen dos posibilidades:

  - $x \geq y$ el máximo es x;
  - $x < y$ el máximo es y;

- La composición secuencial no nos da la posibilidad de tomar decisiones en función de los datos. NECESITAMOS UNA NUEVA COMPOSICIÓN: **composición alternativa, o composición condicional.**

# 1.4 Organización de las acciones: análisis de casos

**LÉXICO**

x, y: Entero; { datos }

z: Entero; { resultado: el máximo de x e y}

**ALGORITMO**

*PRE { x, y : Entero ; x = X, y = Y}*

Leer(x, y);

**SEGÚN** x, y

x ≥ y : z ← x;

x < y : z ← y;

**FIN_SEGÚN;**

Escribir (z)

*POST { x = X, y = Y, y ((z = X) o (z= Y)) y (z ≥ X) y (z ≥ Y) }*

**FIN.**

# 1.4 Organización de las acciones: análisis de casos, composición SEGÚN

**SEGÚN** $c_1, c_2, \ldots, c_n$

$e_1 : a_1$

$e_2 : a_2$

....

$e_m : a_m$

**FIN_SEGÚN**

donde:

$c_i$: nombre perteneciente al dominio del análisis de casos

$e_i$: expresiones booleanas que expresan casos en función de los $c_i$

$a_i$: acción que corresponde a $e_i$

**SEGÚN** $c_1, c_2, \ldots, c_n$

$e_1 : a_1$

$e_2 : a_2$

...

$e_m : a_m$

**EN_OTRO_CASO:** $a_{m+1}$

**FIN_SEGÚN**

$a_{m+1}$ se ejecuta si

$e_1 \vee e_2 \ldots \vee e_m = \text{FALSO}$

# 1.4 Organización de las acciones: análisis de casos, composición SI ENTONCES

- **SI** e **ENTONCES** a **SI_NO** b **FIN_SI;**
- **donde:**
  e: expresión booleana
  a, b: acciones
- Esta composición es equivalente a:

**SEGÚN** $c_1$, $c_2$, ...$c_n$
e : a
**NO** (e) : b
**FIN_SEGÚN;**

- **SI** e **ENTONCES** a **FIN_SI;**

- esta forma es equivalente a:

**SEGÚN** $c_1$, $c_2$, ..$c_n$
**NO** (e) : ; *{ acción vacía }*
e : a
**FIN_SEGÚN;**

# 1.4 Organización de las acciones: análisis de casos. Problema.

- **Problema:** Dados tres enteros diferentes, ordénense de menor a mayor.

- **Especificación:**

  a, b, c, p, s, t: entero

  Precondición { (a=X ^ b=Y ^ c=Z) ^ (X ≠ Y ≠ Z ≠ X) }

  Postcondición { (p, s, t) ∈ *perm*(X, Y, Z) ^ p < s < t }

- **Posibles errores**
  - Olvidar casos
  - Incluir casos que no se excluyan

# 1.4 Organización de las acciones: análisis de casos. Problema.

- En este problema la solución será una de las permutaciones de X, Y, Z por tanto los posibles casos son seis:

| | |
|---|---|
| a<b<c | a<c<b |
| b<a<c | b<c<a |
| c<a<b | c<b<a |

- Hacemos un **análisis de casos** y descomponemos el problema teniendo en cuenta los datos. Utilizamos la composición SEGÚN

- **Datos**: tres variables enteras que representan los datos, **a**, **b** y **c.** Los valores de estas variables deben ser diferentes

- **Resultados**: tres variables enteras, **p**, almacenará el número más pequeño, **s**, el del centro y **t** el mayor

# 1.4 Organización de las acciones: análisis de casos. Problema.

**LÉXICO**
a,b,c: Entero
p,s,t: Entero
**ALGORITMO** **(Solución 1)**
Leer (a, b, c);
**SEGÚN** a, b, c
    a<b<c: p ← a; s ← b; t ← c;
    a<c<b: p ← a; s ← c; t ← b;
    b<a<c: p ← b; s ← a; t ← c;
    b<c<a: p ← b; s ← c; t ← a;
    c<b<a: p ← c; s ← b; t ← a;
    c<a<b: p ← c; s ← a; t ← b;
**FIN_SEGÚN**;
Escribir (p, s, t)
**FIN.**

# 1.4 Organización de las acciones: análisis de casos. Problema.

**ALGORITMO (Solución 2)**

Leer (a, b, c)
**SEGÚN** a,b
a<b: **SEGÚN** a, b, c
b<c: p ← a; s ← b; t ← c;
a<c<b: p ← a; s ← c; t ← b;
c<a: p ← c; s ← a; t ← b;
**FIN_SEGÚN**;
b<a: **SEGÚN** a, b, c
a<c: p ← b; s ← a; t ← c;
b<c<a: p ← b; s ← c; t ← a;
c<b: p ← c; s ← b; t ← a;
**FIN_SEGÚN**;
**FIN_SEGÚN**;
Escribir (p, s, t)
**FIN.**

# 1.4 Organización de las acciones: análisis de casos. Problema.

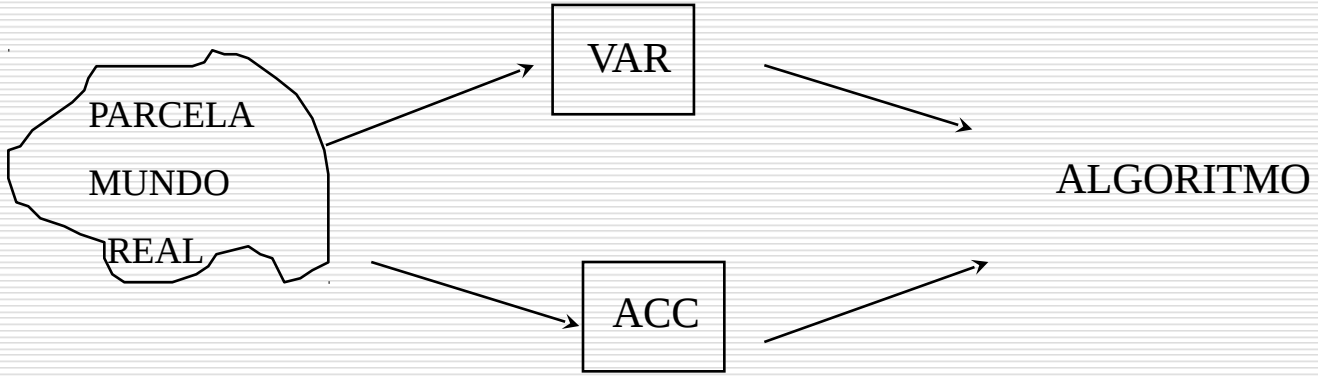
**ALGORITMO (Solución 3)**

p ← a; s ← b; t ← c;
**SEGÚN** p, s, t
p<s y p<t: ; *{ acción vacía }*
s<p y s<t: p ↔ s
t<p y t<s: p ↔ t
**FIN_SEGÚN**;
**SI** s>t **ENTONCES** s ↔ t **FIN_SI**;
Escribir (p, s, t);

**FIN.**

- En este caso la descomposición se ha realizado a partir de los resultados
- Suponemos que disponemos de la operación intercambio de dos variables ↔ (que **no** existe)

# 1.5 Tipos de datos no primitivos




**VAR**: conjunto de variables

**ACC**: conjunto de acciones

La elección de variables influye directamente en la elección de las acciones y viceversa

Los valores del dominio de un tipo pueden ser:

- Atómicos, simples, o escalares

- Estructurados

# 1.5 Tipos de datos no primitivos

- Las notaciones algorítmicas incluyen mecanismos para definir **tipos estructurados.** Los constructores de tipos más usuales son: **tablas, registro (o producto de tipos) y secuencias.**

- **Producto de tipos o registro**:

  Sus valores son una enumeración o agregación de otros tipos ya definidos.
- **Dominio:** n-tuplas de los tipos constituyentes
- **Definición:**

nombre_del_tipo= **TIPO** < $a_1$: $T_1$; $a_2$: $T_2$; ...$a_N$: $T_N$ >

- $a_x$ : denota el nombre de cada uno de los campos o elementos del registro
- $T_1$, $T_2$, ...$T_N$: deben ser tipos ya existentes

# 1.5 Tipos de datos no primitivos

- **Cardinalidad:**
  $C_t = C_{t1} * C_{t2} * .... * C_{tN}$

- **Ejemplo:**
  **LÉXICO**
  Estudiante = **TIPO** < nom: secuencia de carácter;
  edad: entero;
  sexo: booleano;
  centro: entero [1..20];
  curso: entero [1..4] >;
  a1 : estudiante;

- **Posible valor** de la variable a1: <"J. Gómez", 21, Falso, 4, 3>

- **Acceso a elementos**: Operadores **de** y punto:
  curso **de** a1; *o bien* a1.curso