TEMA 2: Parametrización de acciones

2.1 **Parametrización y Léxico local**
- Parámetro y variable local
- Definición y tipos de parámetros
- Ventajas de su uso

2.2 **Funciones**
- Concepto de *función*
- Acciones vs funciones
- Composición funcional vs secuencial

2.3 **Tipo de dato *tabla***
- Características y operaciones
- Tablas vs productos de tipos

Ventajas de la parametrización:

- Permite representar un conjunto potencialmente infinito de diferentes cálculos con un único texto de algoritmo que es una **abstracción** de ellos.
- Es un método para conseguir **generalidad**.
- Definición según la Real Academia Española de:

  Abstraer: separar por medio de una operación intelectual las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción.

  Generalidad: mayoría, muchedumbre o casi totalidad de los individuos u objetos que componen una clase o todo sin determinación de persona o cosa particular.
- Facilitan una escritura de acciones sin referencias a variables del léxico que la incluye: INDEPENDENCIA DEL CONTEXTO EN EL QUE SE INTRODUCE LA ACCIÓN.

Abstracción en acciones parametrizadas:

Doble abstracción:

Abstracción por especificación
Abstracción por parametrización

Ejejmplo:

Intercambio: **una acción** (**dato–resultado** x, y : entero)
{e.i.: x = X, y = Y}
{e.f.: x = Y, y = X}

Especificación vs **implementación**

Localidad: Las acciones parametrizadas pueden ser escritas o comprendidas independientemente del contexto (léxico) en que se utilicen. Su uso implica comprender **sólo su especificación**

Facilitan: la descomposición, legibilidad, mantenimiento, y reutilización

Notación algorítmica para definir acciones

nombre : **una acción** (tp_1 par_1: td_1;....tp_n pa_n: td_n)

donde:

- tp_i: clase de parámetro (*dato*, *resultado* o *dato–resultado*)
- par_i: nombre (identificador) de parámetro
- td_i: tipo de dato del parámetro
- nombre: nombre de la acción

- Definir una nueva acción supone ENRIQUECER el conjunto de acciones primitivas o no primitivas disponibles
- Las acciones son la base para la DESCOMPOSICIÓN DE PROGRAMAS en el paradigma imperativo

Elección de los parámetros de una acción

- Mayor parametrización ⇒ mayor independencia del contexto: **más reutilizable**
- Limitación de la parametrización, implica una adaptación mayor de la acción al contexto la **reutilización directa es menos evidente**

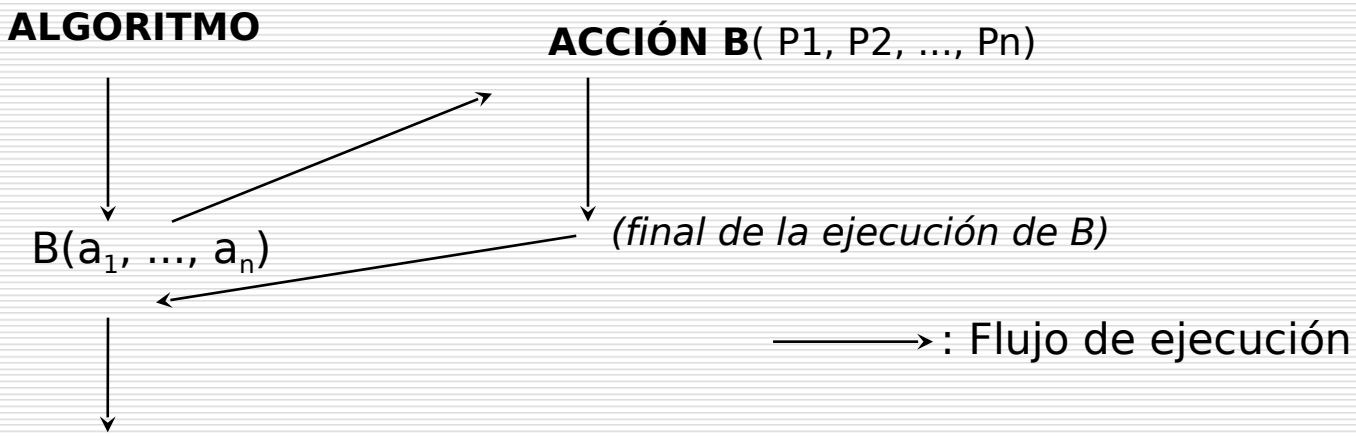
Debemos distinguir entre los parámetros DATO, RESULTADO y los parámetros DATO–RESULTADO.

Parámetros de entrada: valores que se suministran a la acción y no deben modificarse dentro (DATO)

Parámetros de salida: valores calculados por la acción (RESULTADO)

Parámetros de entrada/salida: su valor inicial es necesario para la ejecución de la acción y es modificado por ésta (DATO–RESULTADO)

Abstracción procedural en el uso de una acción



a) Si el parámetro es un DATO de la acción:
(en la llamada) **parámetro ← argumento**
el argumento es una *expresión* del mismo tipo de dato que el parámetro.

b) El parámetro es DATO y RESULTADO de la acción.
(en la llamada) **parámetro ← argumento**
(en el retorno) **argumento ← parámetro**
el argumento es una *variable* del mismo tipo de dato que el parámetro.

Ejemplos de llamadas a acciones y sus efectos en los parámetros y argumentos

LÉXICO

Intercambio: una acción
(dato–resultado x, y : entero)
{ intercambia los valores de x e y }

bin = **TIPO** secuencia de [0,1];
DecBin: una acción (dato m : entero;
dato–resultado b : bin)
{convierte un entero decimal en su equivalente en binario}

ALGORITMO

...

Decbin (x, binario);
DecBin (15, bina);
Intercambio (a, b);
Intercambio (x, b); ...

	LLAMADA	**RETORNO**
	m ← x	binario ← b
	b ← binario	
	m ← 15	bina ← b
	b ← bina	
	x ← a	a ← x
	y ← b	b ← y
	x ← x	x ← x
	y ← b	b ← y

Consideraciones sobre el uso de parámetros

Al emparejarse los parámetros de ambas listas (parámetros formales y argumentos) lo hacen de forma ordenada y deben coincidir en:

- Número de parámetros
- Tipo de parámetros
- Posición en las listas

No es necesario que coincidan los *identificadores* de los argumentos y los parámetros formales:

- Los parámetros de una acción son parte de su *léxico local*
- No son visibles ni utilizables fuera de ella
- Sus identificadores pueden *enmascarar* elementos del léxico superior

Distintos tipos de abstracción en el uso de acciones parametrizadas

ACCIONES ABSTRACCIÓN POR ESPECIFICACIÓN

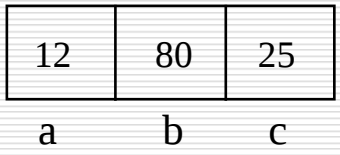
¿Qué hace? vs ¿Cómo lo hace?

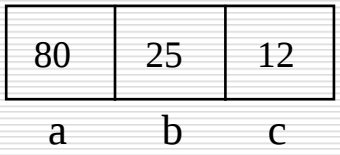
ABSTRACCIÓN POR PARAMETRIZACIÓN

Generalidad vs Especificidad

PARÁMETROS	**ABSTRACCIÓN**
dato	valores de entrada
dato–resultado	objeto

Ejemplo: Desplazamiento circular de los valores de tres variables enteras a, b y c:





Especificación

{ estado inicial: a= A, b= B, c=C }

{ estado final: a= B, b= C, c=A }

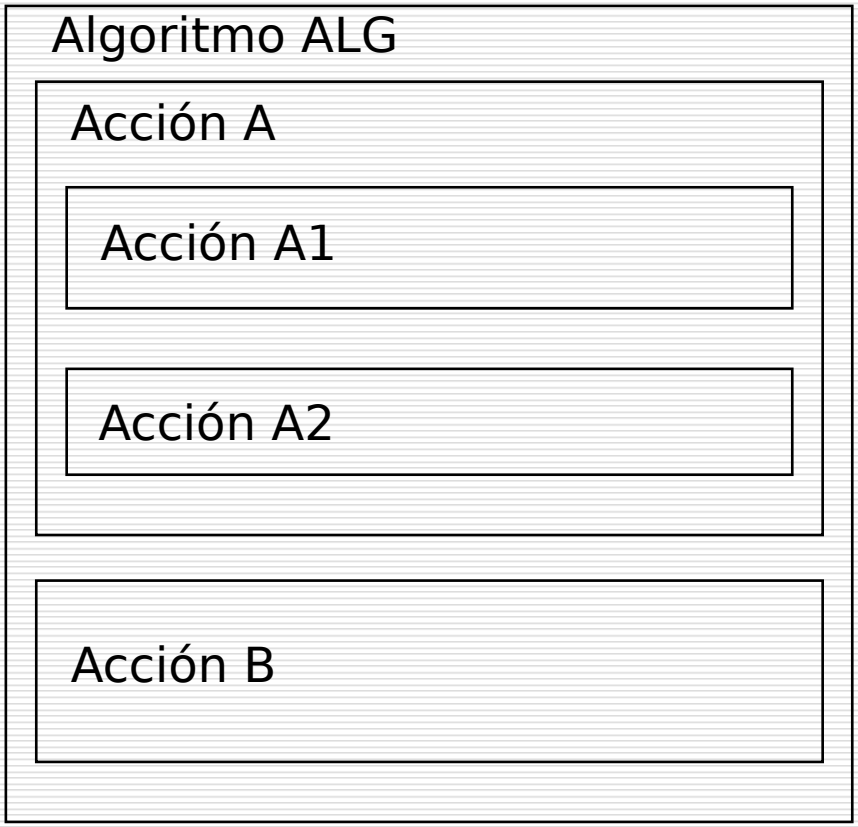
Solución: introducción de un estado intermedio e_1 y de una acción parametrizada para el intercambio de dos variables:

```
LÉXICO
        a, b, c: entero;
        Intercambio : acción (dato–resultado x, y : entero)
        LÉXICO
                t: entero;
        ALGORITMO
                t ← x; x ← y; y ← t
        FIN;
ALGORITMO
        Leer (a, b, c);          { e_i : a=A, b=B, c=C }
        Intercambio (a, b);      { e_1 : a=B, b=A, c=C }
        Intercambio (b, c);      { e_f : a=B, b=C, c=A }
        Escribir (a, b, c)
FIN.
```

variable local a la acción

Niveles de léxico: ámbito y visibilidad de identificadores

Una acción puede contener a su vez otras acciones como parte de su léxico:



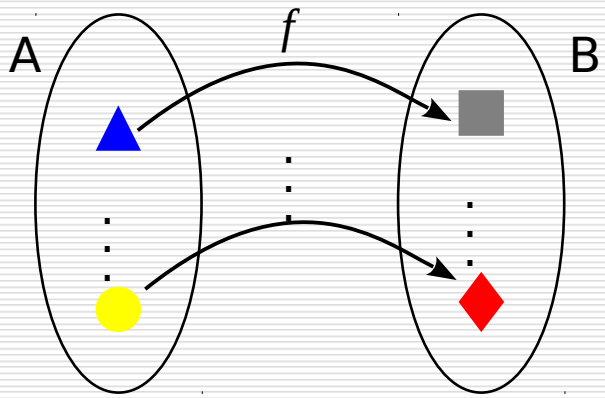
El léxico de:	Tiene ámbito en:
ALG	ALG, A, A1, A2, B
A	A, A1, A2
A1	A1
A2	A2
B	B

Identificadores en léxicos interiores enmascaran otros idénticos en niveles superiores (regla de la *máxima localidad*)

Funciones

Aunque se definen de forma parecida, tienen importantes diferencias con las acciones. En cuanto a su **especificación**:

- **Acciones**: Su efecto es modificar el estado del proceso, en términos de estado inicial y estado final.
- **Funciones**: Establecen una relación entre los elementos de los conjuntos A (dominio) y B (codominio).



$f : A \rightarrow B$

En general el dominio puede estar compuesto por varios conjuntos:

$f : A_1 \times A_2 \times \cdots \times A_n \rightarrow B$

Al igual que las acciones favorecen la **Descomposición** la **Legibilidad** y la **Reutilización**

Funciones

En cuanto a su **utilización**:

- **Acciones**: Describen acciones complejas en función de otras más elementales o primitivas. Se utilizan de idéntico modo a las primitivas.
- **Funciones**: Extienden el repertorio de operadores definiendo otros nuevos. Se utilizan del mismo modo que los operadores, e.d., en expresiones donde pueda aparecer un valor del tipo correspondiente (codominio).

En cuanto a su **descripción**:

- **Acciones**: Su algoritmo establece la forma en que acciones más elementales se combinan para modificar el estado del proceso y conseguir su postcondición a partir de su precondición.
- **Funciones**: También se describen en forma de algoritmo, pero el objetivo de éste no es modificar el estado del proceso sino calcular el elemento del tipo resultado (codominio) que se corresponde con los valores de los parámetros suministrados (dominio).

Funciones: problema ejemplo

Enunciado: Escribir una función que obtenga el valor máximo de tres enteros dados distintos a, b, c.

Especificación:

Max3: ent $\times$ ent $\times$ ent $\rightarrow$ ent

Max3: (a, b, c) =

a	si (a > b) y (a > c)
b	si (b > a) y (b > c)
c	si (c > a) y (c > b)

Funciones: problema ejemplo: ejemplos de uso

Léxico

a, b, c: entero;

Max3: **función** (x, y, z: entero) → entero;

?

Algoritmo

Leer (a, b, c);

.....

t ← a + b + c – Max3 (a, b, c);

....

SI Max3 (a, b, c) = a **ENTONCES** a ← a+1; **FIN_SI**;

.....

Escribir (Max3 (a, b, c));

Funciones: problema ejemplo

SOLUCIÓN 1: Composición funcional:

Introducción de funciones intermedias:

```
Max2: Función (x, y: entero) → entero;
Algoritmo
   Max2 ← ((x+y) + abs (x–y)) DIV 2
FIN;

Max3: Función (x, y, z: entero) → entero;
Algoritmo
   Max3 ← Max2(x, Max2(y, z))
FIN;
```

SOLUCIÓN 2: Análisis exhaustivo de casos:

```
Max3: Función (x, y, z: entero) → entero;
Algoritmo
   SEGÚN x, y, z :
            (x > y) y (x > z) : Max3 ← x;
            (y > x) y (y > z) : Max3 ← y;
            (z > x) y (z > y) : Max3 ← z;
   FIN_SEGÚN
FIN;
```

Funciones: problema ejemplo

SOLUCIÓN 3: Composición funcional con análisis de casos:

```
Max2: Función (x, y: entero) → entero;
Algoritmo
   SEGÚN x, y :
     x > y : Max2 ← x;
     x < y : Max2 ← y;
   FIN_SEGÚN
FIN;

Max3: Función (x, y, z: entero) → entero;
Algoritmo
   SEGÚN x, y :
     x > y : Max3 ← Max2 (x, z);
     y > x : Max3 ← Max2 (y, z);
   FIN_SEGÚN
FIN;
```

SOLUCIÓN 4: Análisis de casos en dos etapas:

```
Max3: Función (x, y, z: entero) → entero;
Algoritmo
  SEGÚN x, y :
    x > y :  SEGÚN x, z :
                x > z : Max3 ← x;
                x < z : Max3 ← z;
             FIN_SEGÚN;
    x < y :  SEGÚN y, z :
                y > z : Max3 ← y;
                y < z : Max3 ← z;
             FIN_SEGÚN;
  FIN_SEGÚN
FIN;
```

Funciones: descripción

- Como puede verse en los ejemplos, una función se describe mediante su propio algoritmo.
- Desde este punto de vista, una función puede visualizarse como una acción en la que:
  - Todos los parámetros son DATO
  - Produce un único resultado que es asignado al nombre de la función como último paso de ésta. El identificador del nombre de la función se considera dentro de ésta como una **variable de sólo escritura**.

Funciones: problema (planteamiento)

Dadas dos duraciones expresadas como <hora, minutos, segundos>, escribir un algoritmo que obtenga la suma expresada de igual modo.

LÉXICO

duración = **TIPO** < h : entero ≥ 0; m, s : [0, 59] >

d1, d2 : duración;

SumarD: **función** (a, b : duración) → duración;

{*PRE*: a, b : duración; a=<H1,M1,S1> b=<H2,M2,S2>}

{*POST*: SumarD=<H1+H2+x, (M1+M2+y) mod 60, (S1+S2) mod 60>

x=(M1+M2+y) div 60

y=(S1+S2) div 60}

// Es preciso definir esta función

ALGORITMO

Leer (d1, d2);

Escribir (SumarD (d1, d2))

Funciones: problema (resolución)

SumarD: **función** (a, b : duración) → duración;
Léxico

función Cvds

Cvds: función (d : duración) → entero;
{duración ⇒ nº ss correspondiente}
Algoritmo // de Cvds
Cvds ← 3600*d.h + 60*d.m + d.s
FIN; // de Cvds

función Cvsd

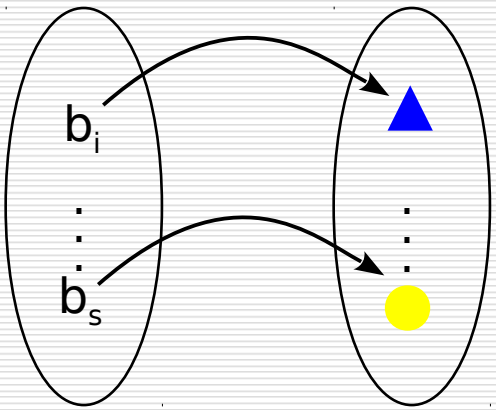
Cvsd: función (n: entero) → duracion;
{nºss ⇒ duracion expresa en hh,mm,ss}
Léxico // de Cvsd
d : duración;
rh : entero [0..3599];
Algoritmo // de Cvsd
d.h ← n div 3600;
rh ← n mod 3600;
d.m ← rh div 60;
d.s ← rh mod 60;
Cvsd ← d
FIN; // de Cvsd

algoritmo de cálculo

composición funcional

Algoritmo // de SumarD
SumarD ← Cvsd (Cvds (a) + Cvds (b))
FIN; // de SumarD

Estructuración de información: tablas



Dominio
(intervalo ordinal)

Rango
(tipo base)

Dominio: intervalo de un tipo ordinal $[b_i, b_s]$
Rango: tipo base cualquiera
b_i, b_s: límites inferior y superior del dominio

Dominio	b_i	b_{i+1}	.....	b_s
Rango	v_1	v_2	.....	v_n

$n = \text{ordinal}(b_s) - \text{ordinal}(b_i) + 1$

OPERACIÓN BÁSICA: acceso directo

TE = **TIPO Tabla** [5, 100] **de** real;
T : TE;

Si $i \in [5, 100]$, T_i representa el elemento de T cuyo índice es i:

Total ← Total + T_i;
T_i ← 30 * s;

Estructuración de información: tablas

OBJETOS	**TABLA**
Temperaturas medias año:	Temp = **TIPO Tabla** [1, 12] **de** real;
Notas en una asignatura:	Mates = **TIPO Tabla** [1, 40] **de** real;
Usuarios del bus en un mes:	BusUsers = **TIPO Tabla** [1, 31] **de** entero;

Diferencias con el producto de tipos:

- Idéntica naturaleza de los elementos
- Nº de elementos *significativos* no fijo
- Elementos referenciados por mecanismo de indexación

OPERACIONES FRECUENTES

- Enumeración secuencial (recorrido): ejecutar una misma acción sobre todos los elementos
- Búsqueda: encontrar un elemento que cumpla cierta propiedad

Tablas: Ejemplo: Diseñar un algoritmo que dada una fecha en formato (día del mes, mes) obtenga el día del año que le corresponde. Ejemplos: (4, 2) → 35, (30, 1) → 30, (31, 12) → 365

Si todos los meses tuviesen 30 días: (a, b) → 30 *(b – 1) + a

Le añadiremos una **correción:** (a, b) → 30*(b – 1) + a + $correc_b$:

mes	1	2	3	4	5	6	7	8	9	10	11	12
correc	0	1	–1	0	0	1	1	2	3	3	4	4

LÉXICO

día = **TIPO** entero [1..31];
mes = **TIPO** entero [1..12];
fecha= **TIPO** <d : día; m : mes>
correc : **TABLA** [mes] **de** entero = [0, 1, –1, 0, 0, 1, 1, 2, 3, 3, 4, 4];
x : Fecha;

ALGORITMO

Leer (x);
Escribir (30 * (x.m – 1) + x.d + $correc_{x.m}$)

FIN.