Bloque II.
2. Objetos y acciones elementales

Objetivos:

– Familiarizarse con conceptos de entorno, objeto y tipo

– Entender qué significa que un objeto sea de un cierto tipo

– Diferenciar los conceptos de constante y variable

– Aprender a utilizar las acciones elementales en lenguaje algorítmico

Bloque II.
2. Objetos y acciones elementales

2.1. Formalización del entorno de un problema

2.1.1. Objetos de un entorno

En las descripciones aparecen: mesas, huevos, folios, sartenes,...

El entorno está formado por **objetos**

***Objeto*: Elemento de un entorno que viene definido por un **nombre** (que lo identifica sin ambigüedad), un **tipo** (que especifica los estados en los que puede estar y las operaciones que pueden realizarse sobre él) y un **valor** (estado concreto en un instante de tiempo).**

Objeto = <nombre, tipo, valor>

2.1. Formalización del entorno de un problema

Ejemplo 1:

tipo mesa :
{escritorio, mesa, {con pila de folios}} o
{escritorio, mesa, {vacía}} o
{mesa número 10, mesa, {vacía}} o
{mesa número 10, mesa, {paquete de folios número 10}}

tipo número:
{media, número, 0.0} o {media, número, 2.5}

Ejemplo 2:

Buscamos convertir pesetas a euros y viceversa, ¿cuál será el entorno?

- **El *estado de un entorno* viene dado por el valor que toman todos los objetos de un entorno en un momento dado.**

2.1. Formalización del entorno de un problema

2.1.2. Constantes y variables

Tienen nombre y tipo definidos.

***Constante*:** Objeto cuyo valor no varía.

Ejemplo: pi, gravedad, cambio_peseta_euro

***Variable*:** Objeto cuyo valor varía a lo largo de la ejecución de un programa.

Ejemplo:
área, velocidad, cantidad_pesetas, cantidad_euros

Un caso especial de variable:

Parámetro: variables cuyo valor permanece constante a lo largo de la ejecución de un algoritmo o programa.

Ejemplo: cambio_euro_dólar

Importante: ¿Valor inicial de un objeto?

Indeterminado

2.2. Tipos elementales

Vienen definidos por:

conjunto de estados (valores) en los que puede estar +
conjunto operadores

***Expresión*:**

secuencia formada por objetos, operadores sobre los objetos y paréntesis

***Tipo de una expresión*:**

el tipo de una expresión viene determinado por el resultado de evaluar la expresión

***Evaluación de una expresión*:**

- se evalúan antes los paréntesis (comenzando por los más internos)
- se evalúan los operadores según el orden de precedencia; en el caso de la misma precedencia, se evaluará de izquierda a derecha

2.2. Tipos elementales

Precedencia entre operadores

nivel 3: - unario, **, NOT
nivel 2: *, /, DIV, MOD, AND
nivel 1: +, -, OR
nivel 0: =, >=, <=, <>, >, <

- **Tipo Numérico**

Estados: números enteros o reales, con o sin signo
Operaciones: aritmética (I.c.i.) y comparación (I.c.e.)

Ejemplo: Supongamos que A, B y C son variables que contienen datos numéricos: ¿Cuál es el tipo de la siguiente expresión?

(3.0 * A) * 75 - 2 * (B - C) > 3 * (A * B - C)

¿Y su valor?

2.2. Tipos elementales

- **Carácter**

Estados: uno y sólo uno de los caracteres alfanuméricos o puntuación, entre comillas simples

Ejemplo: '**a**', '**C**', '**0**', ',', ';', ':', '.'

Operaciones: comparación (I.c.e.)

Ejemplo: ¿Cuál es el tipo de la siguiente expresión? ¿Y su valor?
'a' > '0' '1' < '4'

- **Cadena de caracteres**

Estados: cualquier tira de caracteres entre comillas dobles

Ejemplo: "abc", "casa", "b", "125", "12.5"

Operaciones: comparación (I.c.e.) y concatenación (+: I.c.i.)

Ejemplo: ¿Cuál es el tipo de la siguiente expresión? ¿Y su valor?
"casa" > "cama" "123" > "45" "casa" + " de " + " Pedro"

2.2. Tipos elementales

- **Lógico**

  Estados: cierto o falso

  Operadores: comparación y lógicos (l.c.i.): AND, OR, NOT

  Tipos de expresiones lógicas

  - **Simples: sin operadores lógicos**

    Falso < Cierto

    Cierto < Falso

    A > B

    (2+3) < A

  - **Compuestas: con operadores lógicos**

    (A > B) AND (C < D)

    NOT (C < D)

2.2.2. Declaración de constantes y variables

- **Introducción a la notación BNF**
- **Declaración de objetos:**

  <nombre_constante> = <valor>

  <nombre_variable> {, <nombre_variable>} : <tipo>

  {Ejemplos declaración de constantes}

  pi = 3.14159

  gravedad = 9.8

  cambio_peseta_euro = 166.386

  {Ejemplos declaración de variables}

  pesetas, euros: numérico {reales}

  si_no: lógico

  opción: numérico {entero}

  alfa: cadena

  resp: carácter

2.3. Acción de asignación

2.3.1. Introducción

Inicialmente una variable no tiene valor

La acción de asignación asocia un valor a una variable

Puede ser interna o externa

2.3.2. Asignación interna

Se asocia un valor del entorno a la variable.

<asignación interna> ::=
<variable> := <expresión del tipo de la variable>

Ejemplos:

pesetas := euros * cambio_peseta_euro

alfa := “andrés”

media := (a + b) / 2

2.3. Acción de asignación

2.3.2. Asignación externa

Permite a un algoritmo comunicarse con el exterior.

- Se asocia un valor de fuera del entorno a la variable.
- Se comunica un valor del entorno fuera del mismo.

<asignación externa> ::= <lectura> | <escritura>
<lectura> ::= leer <variable> {, <variable>}
<escritura> ::= escribir <expresión> {, <expresión>}

Ejemplos:

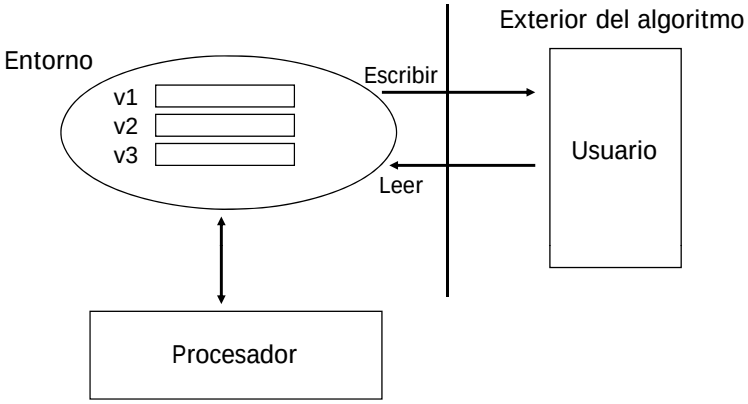
Leer pesetas

Escribir alfa + apellido

Leer euros

Escribir euros * cambio_peseta_euro

Acciones de asignación externa



2.4. Formas de componer acciones

- Asociado al concepto de **programación estructurada**

  Programa propio: aquel que contiene un único punto de entrada y un único punto de salida; además se puede ir desde la entrada hasta la salida recorriendo todo el programa; finalmente, no contiene bucles infinitos.

  Se ha demostrado (Böhm y Jacopini, 1966) que los programas *propios* se pueden realizar exclusivamente mediante esquemas:
  - Secuencial (tema 2),
  - Condicional (tema 4),
  - Iterativo (tema 5)

  <acción>::= <acción-elemental> | <algoritmo> |
  <composición-secuencial-de-acciones> |
  <esquema-condicional> | <esquema-repetitivo>
  <acción-elemental> ::= <asignación-interna> |
  <asignación-externa>

2.4. Formas de componer acciones

- **Composición secuencial de acciones**

  Ejecución incondicional de una secuencia de acciones.
  - Forma más sencilla de componer acciones.
  - Sintaxis:

    <composición secuencial de acciones> ::=
    <acción> {; <acción>}

    <composición secuencial de acciones> ::=
    <acción> |
    <acción> ; <composición secuencial de acciones>

2.5. Descripción de un algoritmo

1. Resumen del problema
2. Principio básico de resolución
3. Descripción del entorno

   Identificar constantes, variables y parámetros.
4. Algoritmo

   Sintaxis:

   Algoritmo [<nombre_algoritmo> [(<parámetros>)]] **es**
   <declaraciones de parámetros, constantes y variables>
   Inicio
   <accion>
   Fin

Ejemplo de descripción de un algoritmo

1. Resumen del problema

   Realícese un algoritmo que permita la conversión de una cantidad en euros a pesetas

2. Principio básico de resolución

   Dado un número de euros, se transforma a pesetas aplicando el tipo de cambio.

3. Descripción del entorno

   Cambio_euro_peseta: numérico;
   {constante; indica el valor del cambio de euros a pesetas; 1 euro = 166'386 pesetas}
   euros: numérico;
   {guarda la cantidad en euros a convertir}
   pesetas: numérico;
   {cantidad en pesetas fruto de la conversión}

Ejemplo de descripción de un algoritmo

4. Algoritmo

```
Algoritmo es
Cambio_euro_peseta = 166.386;
Euros, pesetas: numérico;
Inicio
    Escribir "Convierto euros a pesetas";
    Escribir "Dame la cantidad en euros:";
    Leer euros;
    Pesetas := euros * cambio_euro_peseta;
    Escribir euros, " euros  son ", pesetas, " pesetas";
Fin
```

2.6. Algoritmos con nombre y parametrización de un algoritmo

2.6.1. Algoritmos con nombre

- Los algoritmos anteriores calculan valores y los muestran, pero los resultados de los cálculos no pueden volver a usarse
- Buscamos algoritmos que sean generales:
  calcular con un único algoritmo variantes de un mismo problema
- Los algoritmos son generalizaciones de los operadores básicos: cambio, media, varianza,...
- Estas acciones generalizadas permiten que un mismo algoritmo sea utilizado por otros algoritmos como un operador, mediante una **llamada:**
  solicitar a un procesador que ejecute un algoritmo o acción con nombre

Posibles modificaciones al algoritmo que calcula el cambio

```
Algoritmo cambio es
Cambio_euro_peseta = 166.386;
Euros, pesetas: numérico;
Inicio
    Escribir "Convierto euros a pesetas";
    Escribir "Dame la cantidad en euros:";
    Leer euros;
    Pesetas := euros * cambio_euro_peseta;
    Escribir euros, " euros  son ", pesetas, " pesetas";
Fin
```

Modificaciones:

1. Cambio euro-pesetas o pesetas-euro
2. Seleccionar tipo de cambio
3. Repetir las operaciones

2.6.1. Parámetros de un algoritmo

- Permiten al algoritmo comunicarse con el exterior de su entorno:
  bien con un usuario,
  bien con otros programas
- Hay que distinguir entre:
  - Parámetros formales:
    sintácticos; indican tipo y posición
  - Parámetros reales:
    semánticos; contienen los datos a usar
- Además de los parámetros existen variables internas o locales:
  su valor no puede observarse ni modificarse fuera del algoritmo.

2.6.1. Parámetros de un algoritmo

Ejemplo:
Definición de parámetros formales (sintaxis)

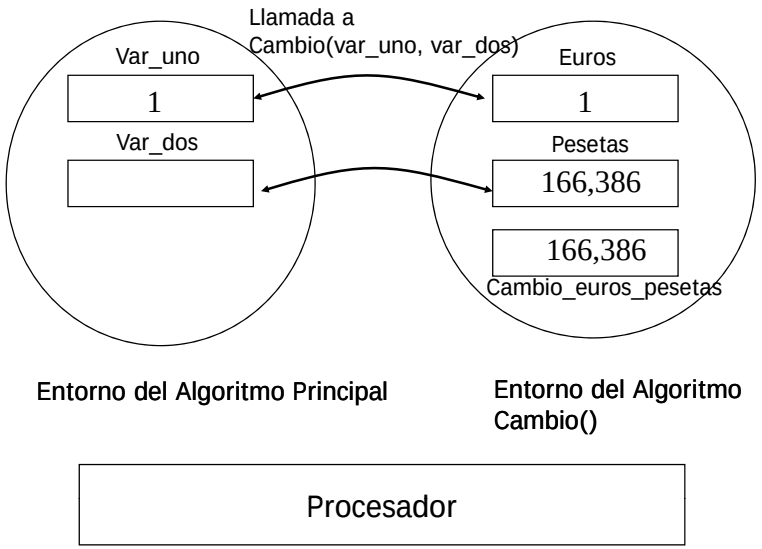
```
Algoritmo cambio (euros, pesetas) es
{declaración de constantes}
cambio_euro_peseta = 166'386;
{declaración parámetros}
euros: numérico; {cantidad de euros a convertir a ptas.}
pesetas: numérico; {resultado de la conversión}
INICIO
    pesetas := euros * cambio_euro_peseta;
FIN
```

2.6.1. Parámetros de un algoritmo

Ejemplo: Uso de parámetros reales (semántica)

```
Algoritmo principal es
{declaración variables}
var_uno, var_dos: numérico;
      {variables en las que se guardan datos}
INICIO
        Escribir "Convierto euros a pesetas";
        Escribir "Dame la cantidad en euros:";
        Leer var_uno;
        cambio (var_uno, var_dos);
        Escribir var_uno, " euros  son ", var_dos, " pesetas";
FIN
```

Paso de parámetros



Paso de parámetros

Entorno del Algoritmo Principal

Procesador

2.6.1. Parámetros de un algoritmo

- Categorías de parámetros:
  - Parámetro dato: no ve modificado su valor
  - Parámetro resultado: no importa su valor inicial
  - Parámetro dato-resultado: usan su valor inicial y lo modifican
- Relaciones con otras nomenclaturas de paso de parámetros:
  - Por valor (se pasa una copia, no se modifican: parámetros dato)
  - Por referencia (se pasa la dirección, pueden modificarse: parámetros dato-resultado)
  - Los valores que devuelve una función (p.e. en C) son parámetros resultado