

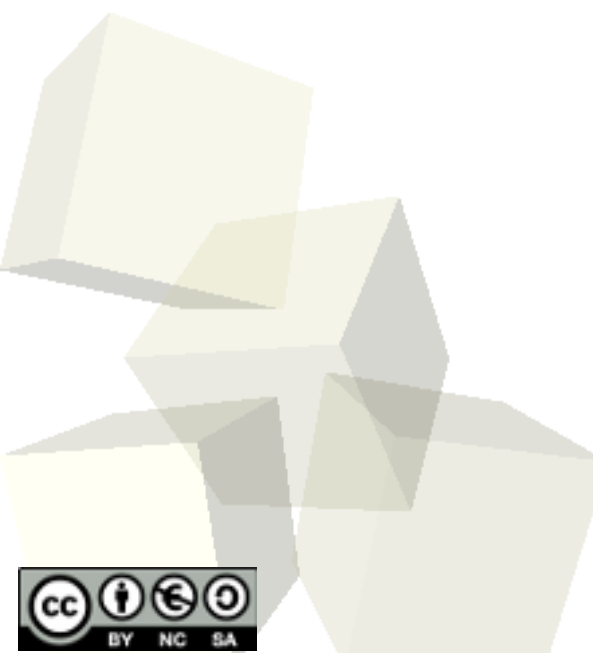


Tema 2

Algoritmos y programas

Informàtica
Grado en Física
Universitat de València

Ariadna.Fuertes@uv.es
Francisco.Grimaldo@uv.es

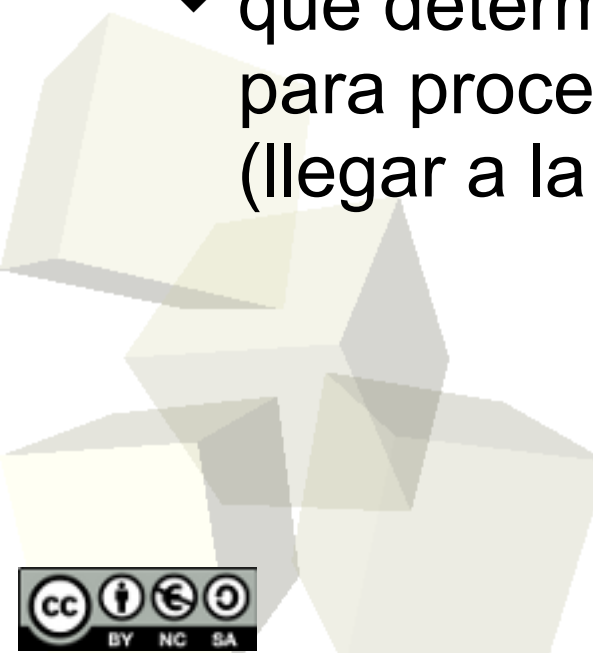




Concepto de algoritmo

■ Un algoritmo es:

- ◆ Una sucesión **finita** de pasos o acciones,
- ◆ especificadas de forma **no ambigua** y
- ◆ que se ejecutan en un **tiempo finito**, y
- ◆ que determinan qué operaciones se deben realizar para procesar datos con un determinado **objetivo** (llegar a la resolución de un problema)





Resolución de un problema

■ Los pasos a seguir son:

1) Análisis del problema:

- Definición del problema.
- Especificaciones de entrada.
- Especificaciones de salida.

2) Diseño/búsqueda del algoritmo:

- Selección/mejora del algoritmo.
- Diseño modular o descendente.
- Refinamiento por pasos.

3) Programación del algoritmo:

- Codificación del programa en:
 - Pseudocódigo.
 - Organigramas o diagramas de flujo.

4) Traducción/ejecución/comprobación del programa.



Ejemplo: Búsqueda números primos

- **Problema:** Buscar los números primos entre 2 y un cierto valor MAX.

1) **Análisis del problema:**

- ◆ ¿Qué es un número primo? ¿Qué entradas tenemos? ¿Qué queremos obtener como resultado?

2) **Búsqueda del algoritmo:**

◆ Método 1:

1. $X = 2$
2. $I = 2$
3. Hacer (X/I)
4. Si I es menor que X y la división es entera entonces **X no es primo** y pasar a 7
5. Si I es igual que X entonces **X es primo** y pasar a 7
6. Incrementar I y pasar a (3).
7. Si X es más pequeño que MAX entonces incrementar X y pasar a (2).
8. FIN

Mejora del algoritmo: parar la división si la ' I ' es mayor que ' $X/2$ '

◆ Método 2: Criba de Eratóstenes

1. Poner todos los números entre 2 y MAX uno detrás de otro.
2. Si hay números sin tachar, el primero de ellos es primo
3. Tachar de la lista todos los múltiplos del primer número
4. Borrar el primer número
5. Borrar los tachados y pasar a (2).

3) **Programación** + 4) **Traducción/ejecución/comprobación**

Ejemplo: Suma sucesión aritmética

■ **Análisis del problema:**

- ◆ Descripción del problema: $a_1 + a_2 + a_3 + \dots + a_n$ donde $a_2 = a_1 + d$ y en general $a_n = a_1 + (n-1) \cdot d$
- ◆ Entradas: Primer término (a_1)
 Distancia (d)
 Número de términos (n)
- ◆ Salida: Suma de todos los términos. $\sum_{i=1}^n a_i = S_n$

■ **Búsqueda del algoritmo:**

- ◆ **Método 1:** Calcular los términos e ir sumándolos poco a poco.

1. $S_n \leftarrow 0$
2. $i \leftarrow 1$
3. $X \leftarrow a_1$
4. Si i es más grande que n , entonces saltar a (9).
5. $S_n \leftarrow S_n + X$
6. $X \leftarrow X + d$
7. $i \leftarrow i + 1$
8. Volver a (4).
9. Mostrar el resultado (S_n).
10. FIN.

- ◆ **Método 2:**

1. Aplicación de la fórmula:
2. Mostrar el resultado (S_n).
3. FIN.

$$S_n = n \cdot a_1 + \frac{n \cdot (n-1)}{2} \cdot d$$



■ Partes importantes en la realización de un programa:

1.- Funcionamiento del programa

2.- Claridad

- indentado, sangrías
- comentarios (al usuario y al corrector).

3. **Estilo** de programación:

- buen y correcto uso de variables
- algoritmos usados
- uso de funciones y procedimientos.



Ejemplo: Múltiplos de cinco

■ Problema:

- ◆ Calcular la cantidad de múltiplos de cinco que existen entre 0 y un n° introducido por teclado.

■ Método 1:

1. $DIV \leftarrow 5$
2. Leer (MAX)
3. $I \leftarrow 0$
4. Si DIV es mayor que MAX, ir a (8).
5. $I \leftarrow I + 1$
6. $DIV \leftarrow DIV + 5$
7. Ir a (4).
8. Mostrar (I).
9. FIN.

■ Método 2:

1. $DIV \leftarrow 5$
2. Leer (MAX)
3. $I \leftarrow$ parte entera de MAX/DIV
4. Mostrar (I).
5. FIN.



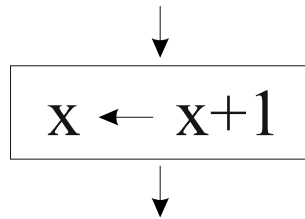
Programación del algoritmo

- El **pseudocódigo** es una manera de escribir algoritmos de forma poco estricta (con una sintaxis relajada) o estructuras de datos poco detalladas, pero intentando acercar las ideas del algoritmo a estructuras y sintaxis parecidas a las de los lenguajes de alto nivel en los que vamos a programar el algoritmo.
- Los **organigramas** o **diagramas de flujo** son dibujos que representan de manera gráfica tanto las tareas como la sucesión de tareas del algoritmo. Las tareas se representan mediante rectángulos, rombos y romboides y el flujo de tareas mediante flechas que enlazan las diferentes tareas.

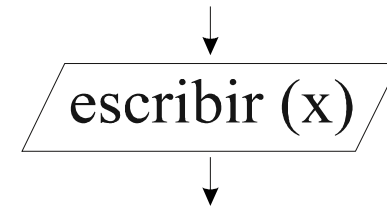


Diseño de Organigramas

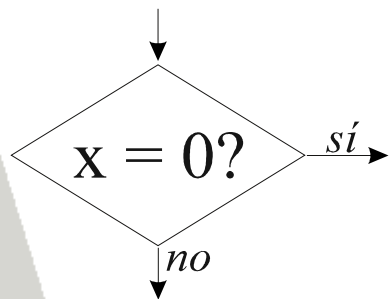
- Las **instrucciones** se representan en rectángulos:



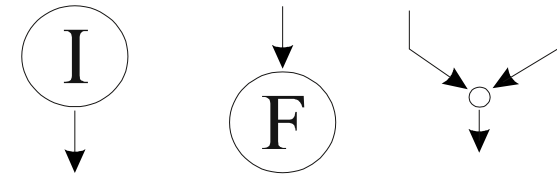
- Las **entradas y salidas** en romboides :



- Las **condiciones** en rombos:



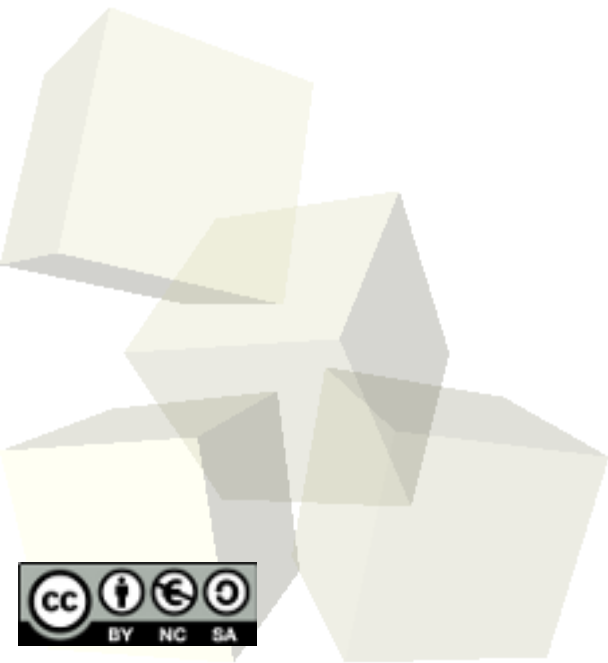
- El **inicio**, el **final** y los **puntos de reunión** de flujo en círculos:





■ Problema:

- ◆ Escribir un algoritmo que calcule los cuadrados de los números enteros positivos hasta que el cuadrado sea mayor o igual que 100.
- Representar el algoritmo anterior mediante organigramas.





Concepto de variable/constante

- **Variable**: dirección de memoria cuyo contenido puede cambiar durante la ejecución de un programa
- **Constante**: dirección de memoria cuyo contenido permanece constante durante la ejecución de un programa.





Ámbito de las variables

- **Variable local:** Son propias de cada módulo y sólo existen mientras dura la ejecución del módulo. Cuando la ejecución del módulo desaparece, las variables locales desaparecen
- **Variable global:** Son variables que existen durante toda la ejecución del algoritmo, de manera que se puede acceder a ellas en cualquier momento de la ejecución del algoritmo



Estructuras de control

- Llamaremos estructuras de control a las acciones que tienen como objeto marcar el **orden de ejecución** de las instrucciones y que van a servirnos para escribir concisamente y sin ambigüedades los algoritmos.
- Todas las estructuras de control que estudiaremos estarán compuestas de unos elementos básicos (léxico) y una estructura (sintaxis.).
- **Tipos:** Secuenciales, Selectivas, Repetitivas.



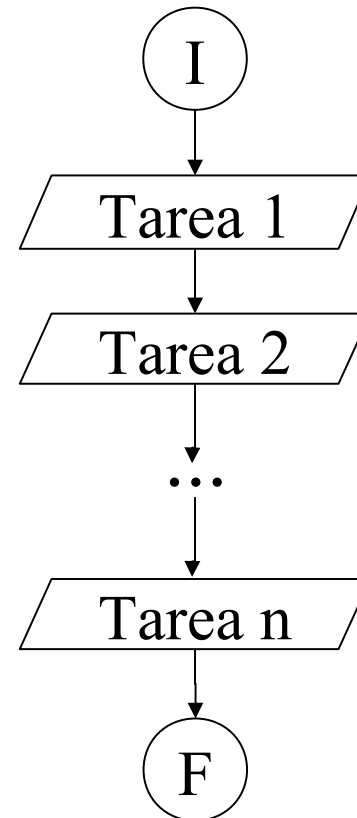
Estructuras Secuenciales

- En una estructura secuencial una instrucción sigue a otra en una secuencia lineal:

Pseudocódigo:

Inicio
Tarea 1
Tarea 2
...
Tarea n
Fin

Organigrama:





Ejemp. de estructuras secuenciales

- Producto escalar de dos vectores bidimensionales:
 $(ax, ay) (bx, by) = ax * bx + ay * by$

Pseudocódigo:

Inicio

Leer (**ax**)

Leer (**ay**)

Leer (**bx**)

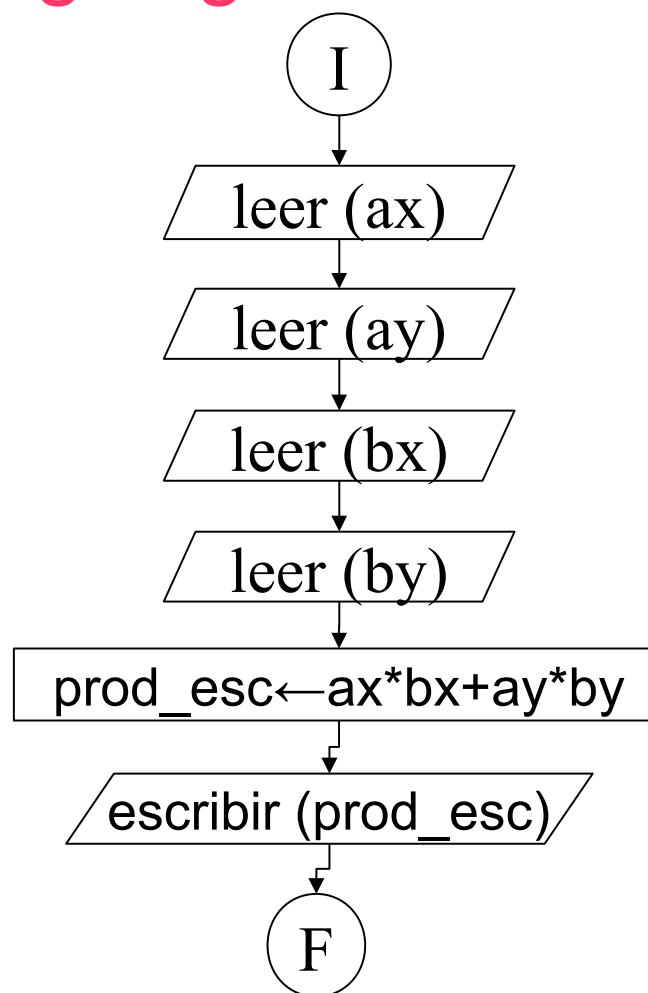
Leer (**by**)

prod_esc ← **ax*bx+ay*by**

Escribir (**prod_esc**)

Fin

Organigrama:





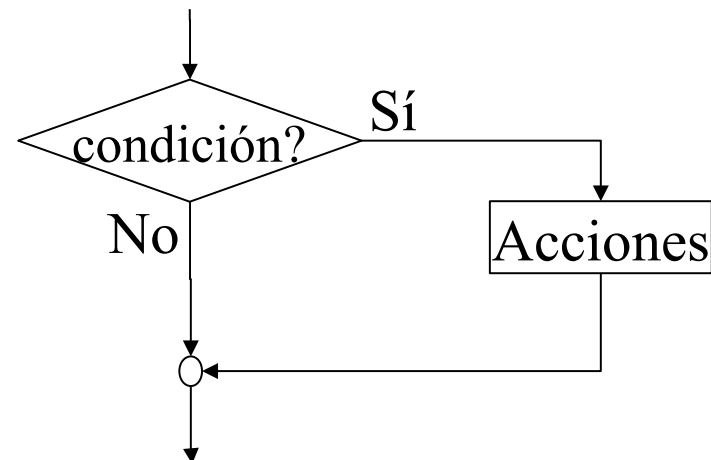
Estructuras selectivas

- Son las que toman una cierta dirección dentro del flujo del programa en función de una condición o el valor de una variable.
- Alternativas simples:
 - ◆ Se realiza una acción o conjunto de acciones si se cumple una determinada condición.

Pseudocódigo:

...
Si (condición) entonces
 Acciones
Fin_si

Organigramas:





Ejemplo de selectivas simples

- Ordenar dos números leídos por teclado.

Pseudocódigo:

Organigrama:

Inicio

Leer (a)

Leer (b)

Si (a > b) entonces

aux ← a

a ← b

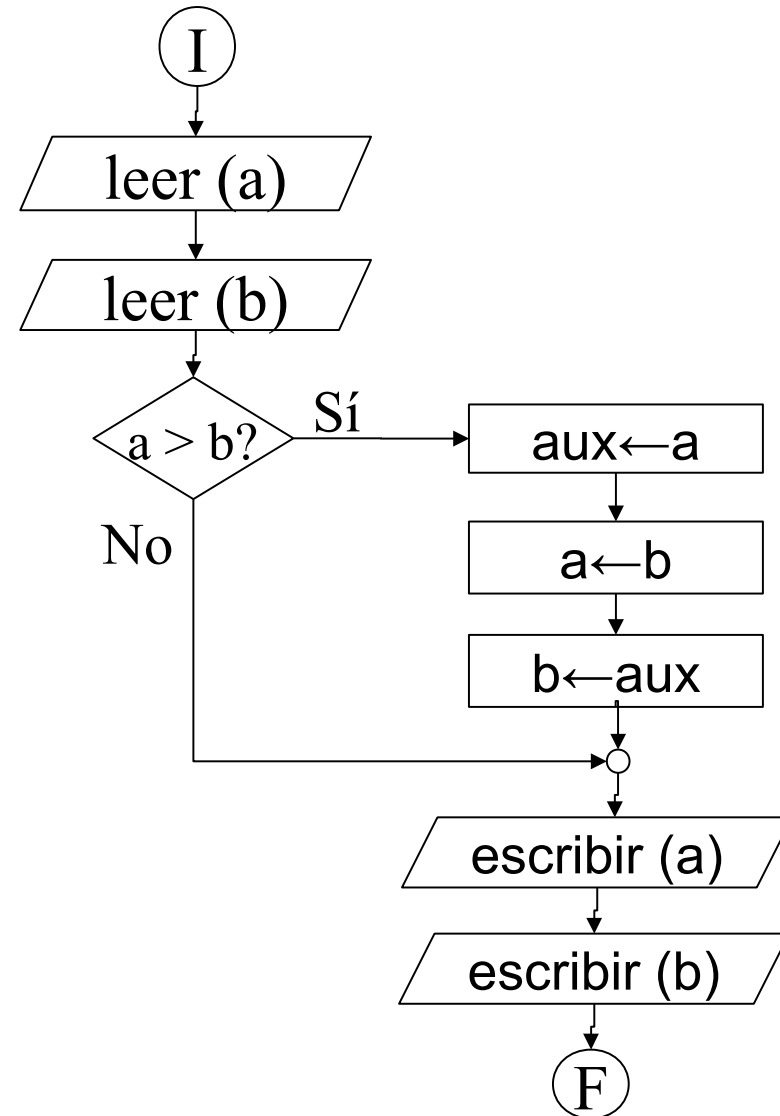
b ← aux

Fin_si

Escribir (a)

Escribir (b)

Fin





Estructuras selectivas dobles

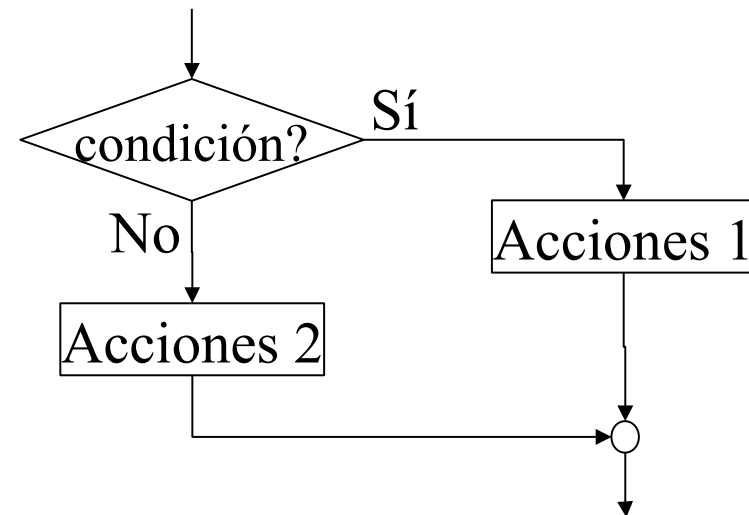
■ Alternativas dobles:

- ◆ Si una condición se cumple se realizan unas acciones, si no se cumple la condición se realizan otras.

Pseudocódigo:

...
Si (condición) entonces
 Acciones 1
Sino
 Acciones 2
Fin_si

Organigrama:





Ejemplo de selectivas dobles

- Dado un número, decir si es positivo o no.

Pseudocódigo:

Variables

x: Entero

Inicio

Leer(**x**)

Si (**x** < 0) entonces

 Escribir("Negativo")

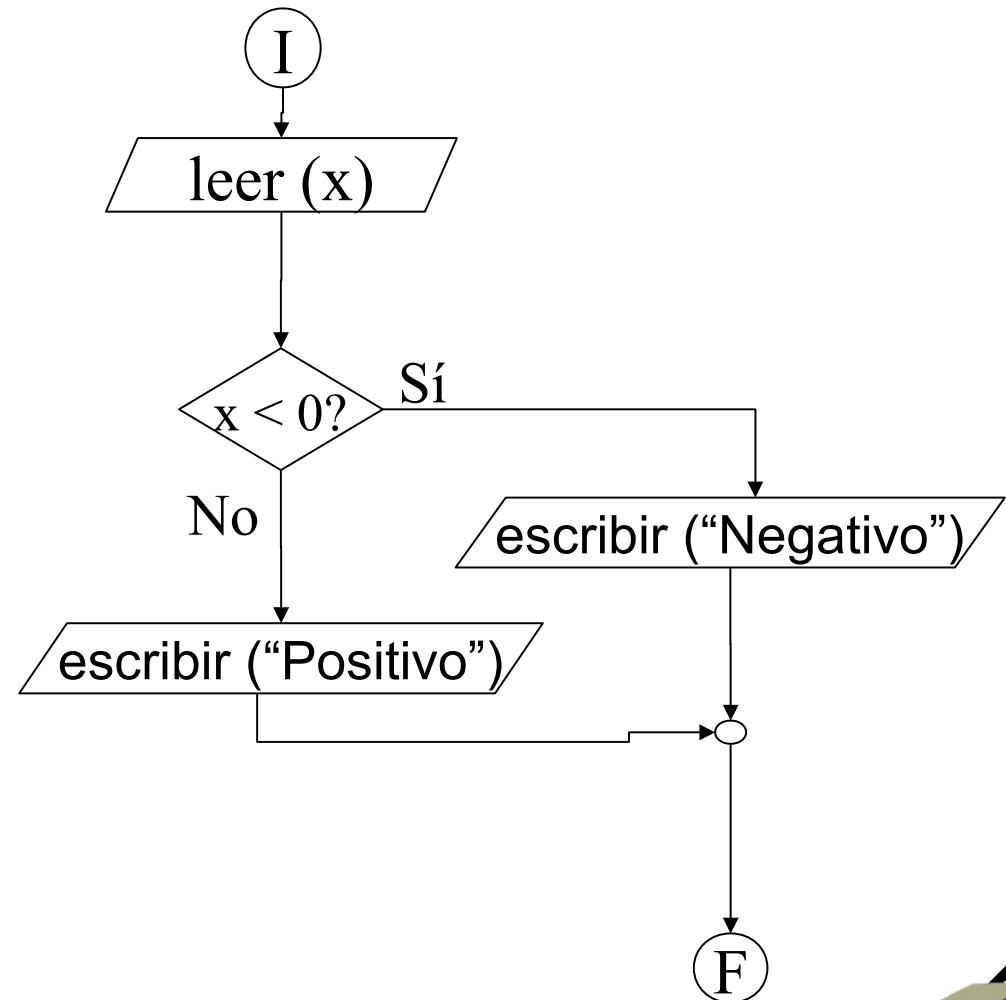
Si no

 Escribir("Positivo")

Fin_si

Fin

Organigrama:



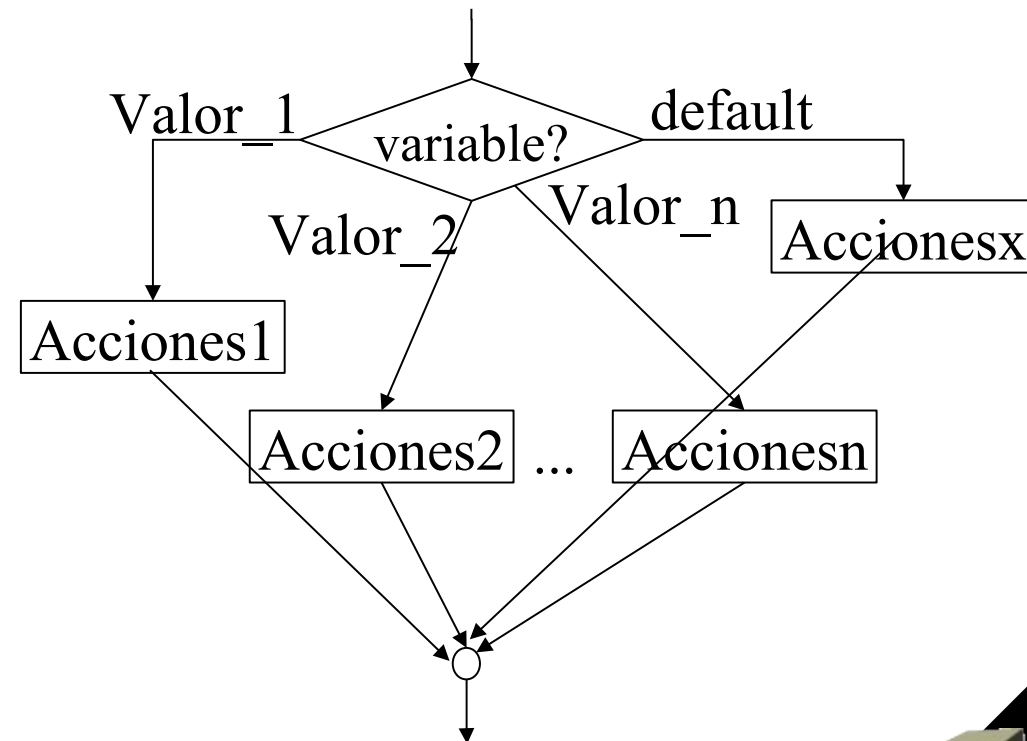
Estructuras selectivas múltiples

- Alternativas múltiples:
 - ◆ Dependiendo del valor de una variable se realizan unas acciones u otras.

Pseudocódigo:

...
Según_sea (**variable**) hacer
 Caso **valor1**: Acciones1
 Caso **valor2**: Acciones2
 ...
 Caso **valorn**: Accionesn
 Default: Acciones x
Fin_segúns_sea

Organigrama:



Ejemplo de selectivas múltiples

- Realizar un programa que pida dos números y una operación (suma, resta, multiplicación o división) y nos de el resultado de operar los números con esa operación.

Pseudocódigo:

Inicio

Escribir ("Dame números?")

Leer (x,y)

Escribir ("Operació?")

Escribir ("1-Suma, 2-Resta")

Escrebiriure ("3-Multi, 4-Divisio")

Leer (op)

Según_sea (op) hacer

Caso 1: $res \leftarrow a + b$

Escribir (res)

Caso 2: $res \leftarrow a - b$

Escribir (res)

Caso 3: $res \leftarrow a * b$

Escribir (res)

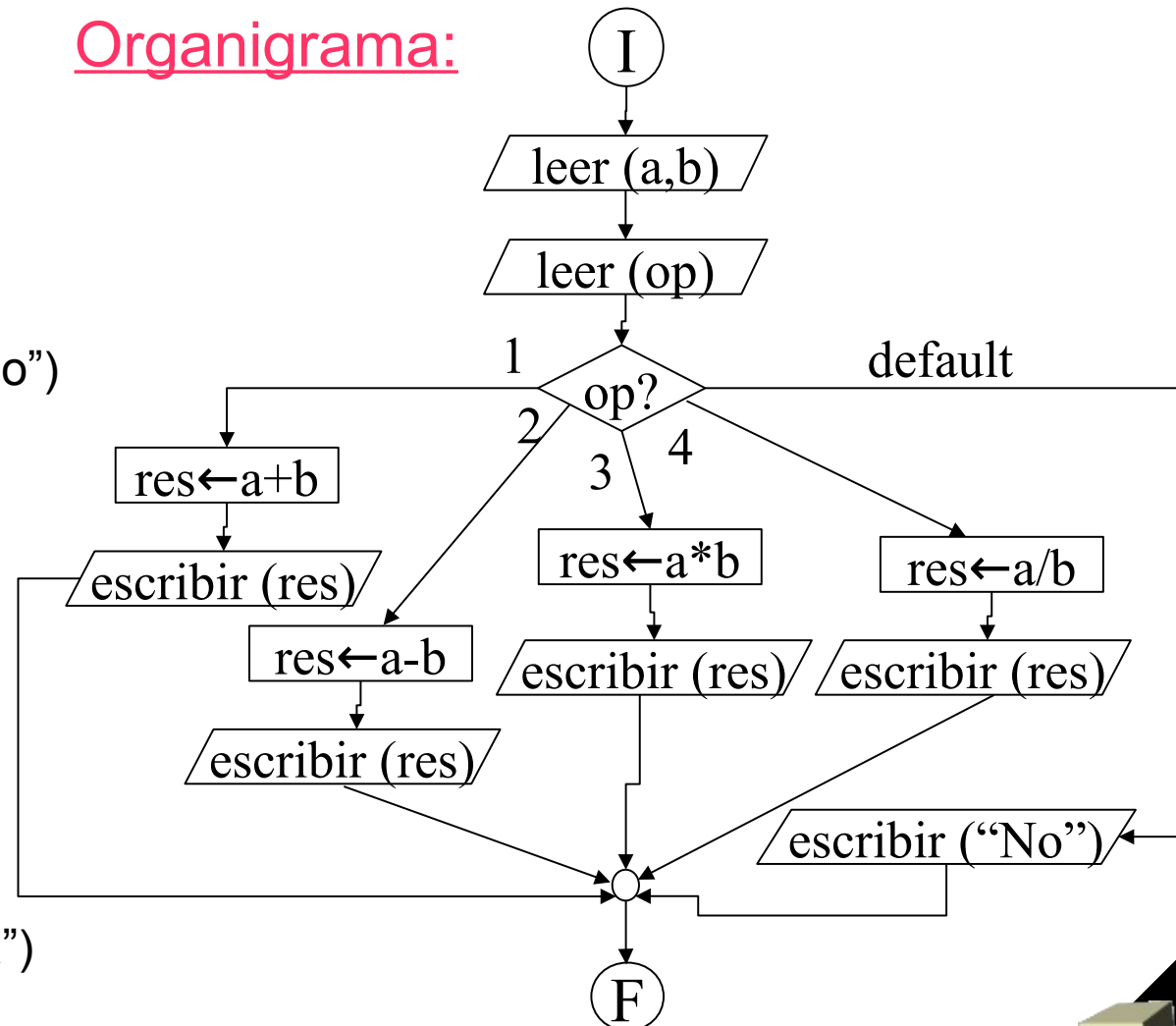
Caso 4: $res \leftarrow a / b$

Escribir (res)

Default: Escribir ("Op. No válida")

Fin_según_sea

Organigrama:



Estructuras repetitivas (bucles)

- Un bucle es un conjunto de **instrucciones del programa que se ejecutan repetidamente** o bien un número determinado de veces, o bien mientras se cumpla una determinada condición (hay que tener cuidado con los bucles infinitos).
- Todo bucle contiene los siguientes **elementos** (aunque no necesariamente en ese orden):
 - Iniciación de las variables referentes al bucle.
 - Decisión (seguimos con el bucle o terminamos.)
 - Cuerpo del bucle
- Existen tres **tipos** de bucles:
 - ◆ Desde ... hasta
 - ◆ Hacer ... mientras
 - ◆ Mientras ... hacer.



Bucle Desde ... Hasta

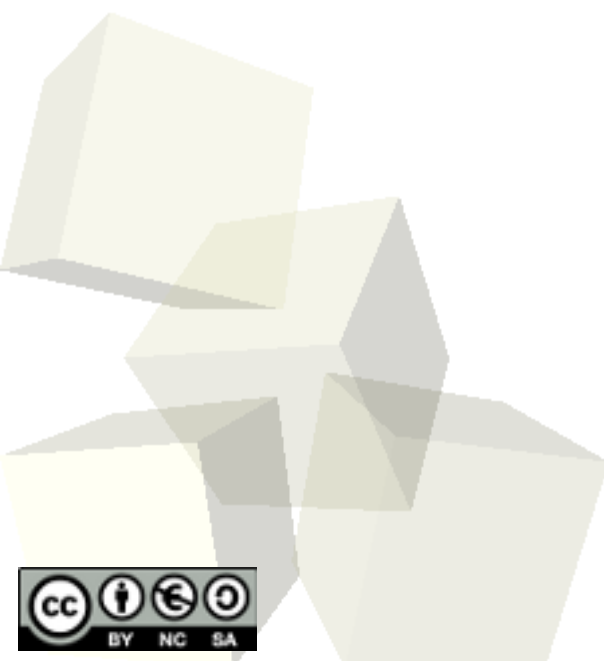
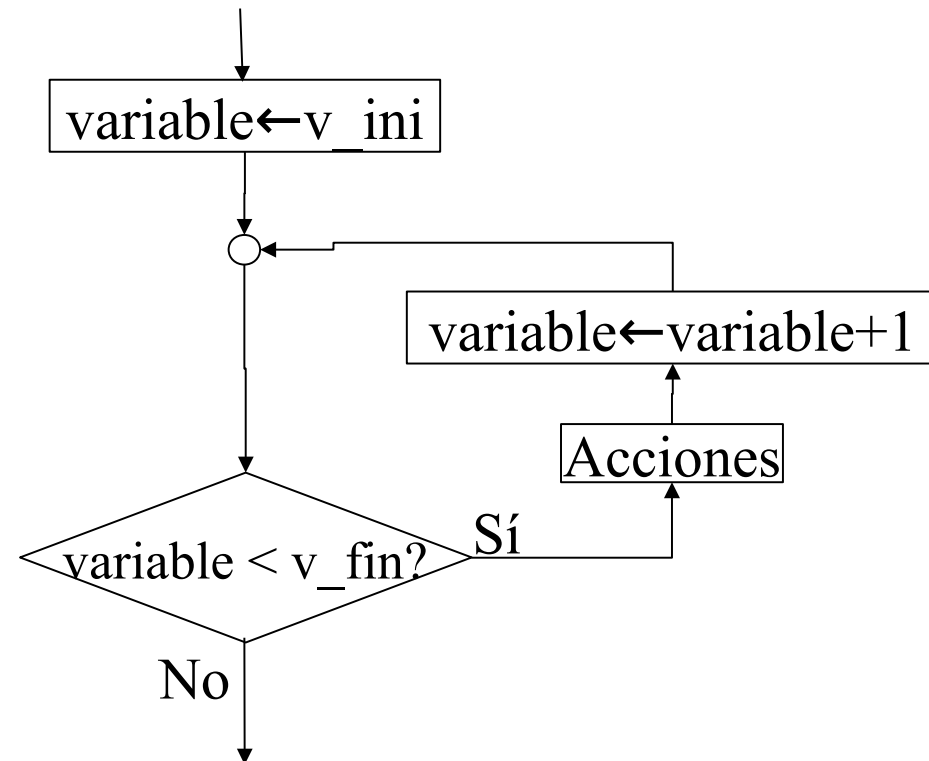
- Cuando sabemos el número de veces que queremos que se realice una tarea.

Pseudocódigo:

...
Desde **variable** ← v_ini hasta v_fin hacer
 Acciones
Fin_Desde

...

Organigrama:





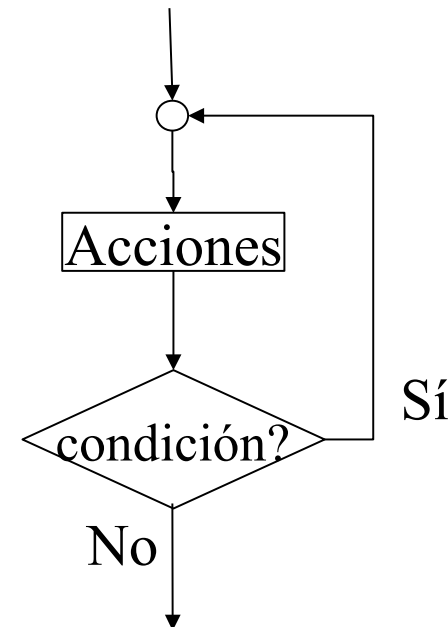
Bucle Hacer ... Mientras

- Si sabemos la condición que hace que se repita la tarea varias veces.
- Las acciones se realizan al menos una vez, antes de realizar la comprobación de la condición.

Pseudocódigo:

...
Hacer
 Acciones
Mientras (condición)
...
...

Organigrama:





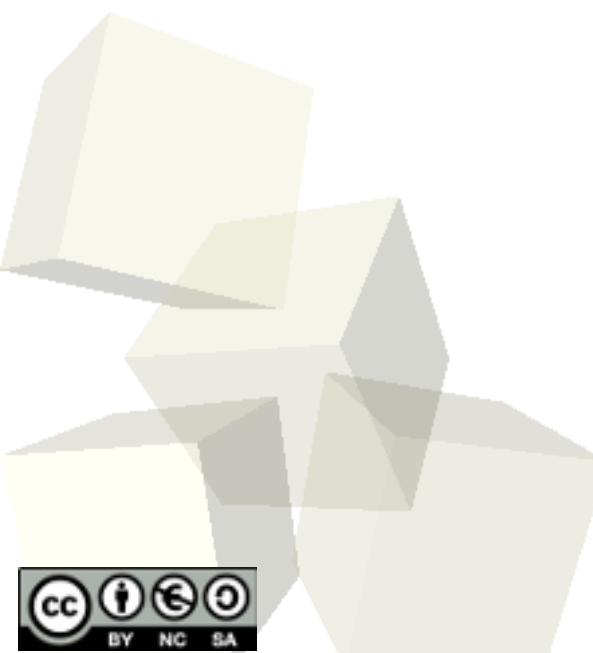
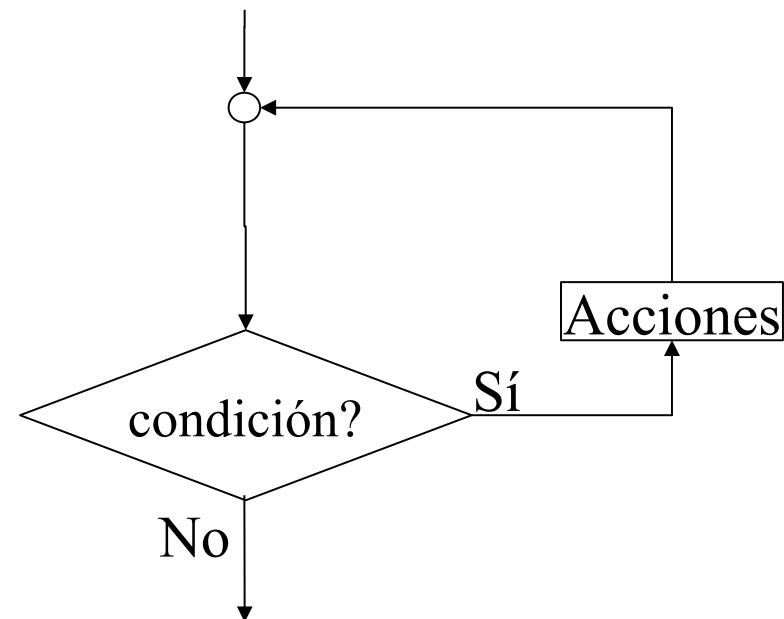
Bucle Mientras ... Hacer

- Parecido al anterior pero la comprobación se realiza antes de realizar la tarea

Pseudocódigo:

...
Mientras (condición)
 Acciones
Fin_Mientras
...

Organigrama:



Bucles independientes o anidados

- Existen dos maneras básicas de utilizar varios bucles: de forma anidada y de forma independiente
 - ◆ **Bucles independientes:** Cuando usamos los bucles uno detrás de otro. Es decir, nos limitamos a ir haciendo los bucles de manera que al finalizar uno empezará el siguiente y de esta forma, las tareas entre los bucles son independientes.
 - ◆ **Bucles anidados:** Son bucles que están dentro de otros bucles de manera que la ejecución de los bucles internos depende de la ejecución de los bucles externos.



Programación modular

- Las características básicas que deben cumplir los **subprogramas** o **módulos** son:
 - a) Realización de una tarea específica.
 - b) Parametrización para caracterizar la actuación de los módulos (parámetros).
 - c) Existen básicamente dos tipos distintos de subprogramas:
 - **Las funciones:** devuelven el resultado de su actuación mediante un único valor (o más, si recibe parámetros por referencia).
 - **Los procedimientos:** no devuelven ningún valor (aunque pueden devolver diversos resultados mediante los parámetros por referencia).



Paso de parámetros a módulos

■ Por valor:

- ◆ Sólo se tiene en cuenta el valor del parámetro pasado al módulo, de manera que durante la ejecución del módulo se reserva un espacio para ese parámetro y se copia el valor pasado en ese nuevo espacio.
- ◆ Cuando acaba la ejecución del módulo, el espacio para el parámetro desaparece.

■ Por referencia:

- ◆ Lo que pasamos a la función es una referencia al parámetro que se pasa, de manera que no existe un espacio reservado para ese parámetro durante la ejecución del módulo.
- ◆ Cualquier modificación que realicemos del parámetro quedará reflejado en el punto desde el que se hizo la llamada al módulo.



Programación estructurada

- Diremos que estamos realizando una **programación estructurada** cuando:
 - ◆ Empleamos para el diseño de los algoritmos el diseño descendente y/o modular.
 - ◆ Descomponemos, en función del diseño descendente, el programa en módulos independientes (prohibida la utilización de variables globales.).
 - ◆ En cualquier caso, sólo utilizamos en la escritura de los algoritmos (y los programas) los tres tipos de estructuras de control vistas.



Recurrencias o recursividad

- Es la propiedad de llamar a una función desde sí misma o desde otra que ha sido llamada por ésta.
- Una característica muy importante de la recursividad es que, igual que en las estructuras repetitivas, debe de existir **un punto o condición de fin**, que en algún momento detenga la recursividad.

