



TEMA 01 - 02

DIAGRAMAS DE FLUJO

PROGRAMACIÓN
CFGS DAM

2022/2023

Versión: 230913.1658

Autores: Carlos Cacho y Raquel Torres


Revisado por:


Lionel Tarazón - lionel.tarazon@ceedcv.es


Fco. Javier Valero – franciscojavier.valero@ceedcv.es

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 Importante

 Atención

 Interesante

ÍNDICE DE CONTENIDO

1. INTRODUCCIÓN.....	4
2. INSTRUCCIONES DE INICIO Y FIN.....	4
3. INSTRUCCIONES DE PROCESADO DE INFORMACIÓN.....	4
4. INSTRUCCIONES DE ENTRADA Y SALIDA DE INFORMACIÓN.....	6
5. ESTRUCTURAS DE CONTROL.....	9
6. ESTRUCTURAS ALTERNATIVAS.....	9
6.1 Estructura alternativa simple.....	10
6.2 Estructura alternativa doble.....	11
6.3 Estructura de alternativa múltiple.....	12
7. Introducción.....	13
8. Estructura Mientras (WHILE).....	13
9. Estructura Hasta (DO-WHILE).....	14
10. Estructura Para (FOR).....	15
11. Formas de acabar un bucle.....	16
12. Elementos auxiliares.....	17
12.1 Contadores.....	17
12.2 Acumuladores.....	17
12.3 Interruptores.....	18
13. Licencia.....	18
13.1 Agradecimientos.....	18

DIAGRAMAS DE FLUJO

1. INTRODUCCIÓN

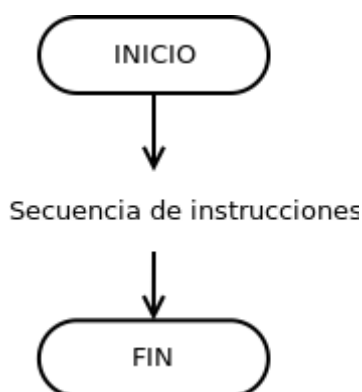
Como ya vimos en la unidad anterior, los **diagramas de flujo** (u ordinogramas) son una forma gráfica de **representar algoritmos**. Dicho de otro modo, **representa las instrucciones paso a paso que se ejecutarían en un programa de ordenador**.

Son muy útiles para practicar y aprender a pensar de forma algorítmica antes de programar utilizando lenguajes de programación reales como Java, Python, etc. Los ordinogramas pueden crearse directamente con papel y lápiz o utilizar algún software como [DIA](#), [yEd](#) o [draw.io](#).

Veamos uno a uno los diferentes elementos de un ordinograma y cómo utilizarlos.

2. INSTRUCCIONES DE INICIO Y FIN

Todos los programas tienen un punto inicial y un punto final que marcan cuándo empieza y acaba un programa. En un ordinograma estos se representa mediante **INICIO** y **FIN**.



3. INSTRUCCIONES DE PROCESADO DE INFORMACIÓN

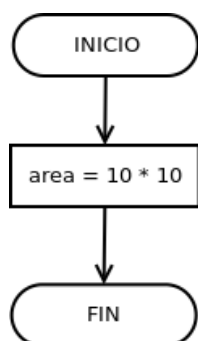
Todos los programas informáticos procesan información, es decir, realizan operaciones matemáticas para calcular, manipular o modificar información. Como ya vimos en la unidad anterior las operaciones pueden ser aritméticas (+ - * / %), relacionales (< <= > >= == <>) o lógicas (NOT, AND, OR). A su vez, se utilizan variables (A, B, Edad, Precio, etc.) para almacenar la información que estamos procesando.

El encargado de realizar este tipo de operaciones es el procesador de un ordenador.

En un ordinograma se representan utilizando **rectángulos** que contienen las instrucciones.

Por ejemplo, el siguiente ordinograma calcula el área de un cuadrado de lado 10.

ORDINOGRAMA

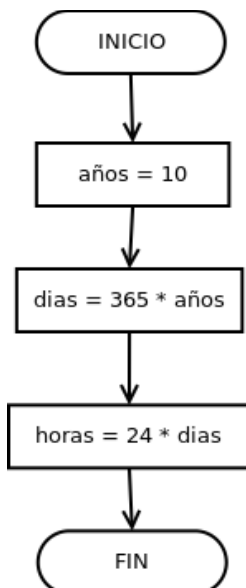


EXPLICACIÓN

Calcula $10 * 10$ y guarda el resultado (100) en la variable **area**.

Podemos utilizar tantas instrucciones de procesamiento de información como necesitemos. Por ejemplo, el siguiente ordinograma calcula el número de horas que hay en 10 años.

ORDINOGRAMA



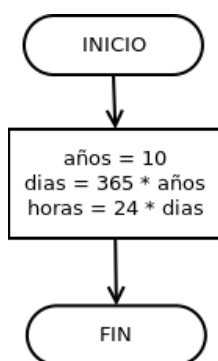
EXPLICACIÓN

Guardamos 10 en la variable **años**.

Calcula $365 * \text{años}$ ($365 * 10$) y guarda el resultado (3650) en la variable **días**.

Calcula $24 * \text{días}$ ($24 * 3650$) y guarda el resultado (87600) en la variable **horas**.

Si se desea, es posible poner varias instrucciones dentro de un mismo rectángulo (para simplificar):



4. INSTRUCCIONES DE ENTRADA Y SALIDA DE INFORMACIÓN

Todos los programas informáticos utilizan algún tipo de entrada y salida de información. Es una parte muy importante de la programación ya que, entre otras cosas, es lo que permite a un usuario interactuar con un programa. Por ejemplo:

1. El usuario introduce información por teclado (entrada).
2. El programa procesa la información (hace algún cálculo).
3. El programa muestra el resultado por pantalla (salida).

Todos los programas de ordenador, apps de teléfono, páginas web, etc. siguen estos tres pasos. Tú aprietas un botón (o haces click, tocas la pantalla...), luego el procesador lo procesa y por último sucede algo (se visita una página web, se escucha una canción, se envía un whatsapp, etc.).

No tendría sentido crear programas sin entrada ni salida ya que solo realizarían cálculos sin comunicarse con el exterior (como en los dos ejemplos del apartado anterior).

Algunos ejemplos de dispositivos utilizados para entrada y salida de información:

- Dispositivos de Entrada: teclado, ratón, micrófono, escáner, gps, wifi, etc.
- Dispositivos de Salida: pantalla, altavoces, impresora, wifi, etc.

Por ahora nos vamos a centrar en la entrada y salida más sencilla: el teclado y la pantalla.

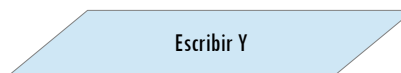
En los ordinogramas la entrada y salida de información se representa mediante un **paralelogramo**.

ENTRADA de información



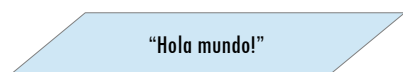
El usuario introduce información por teclado y se guarda en la variable X.

SALIDA de información

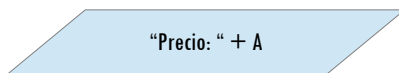


Muestra por pantalla el contenido de la variable Y.

La instrucción de SALIDA de información es muy versátil. También puede utilizarse para mostrar texto, además de combinaciones de texto y variables.



Muestra por pantalla el texto "¡Hola mundo!"

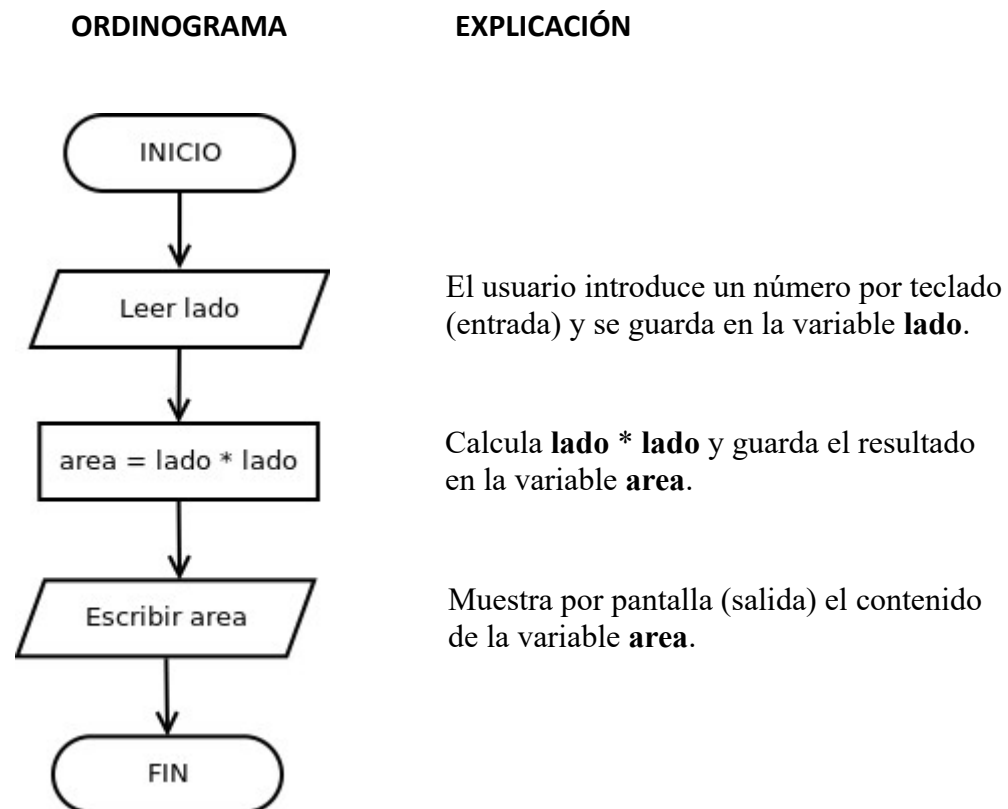


Muestra por pantalla el texto "Precio : " seguido del valor de la variable A.

Si por ejemplo A vale 15, mostrará el texto "Precio: 15".

¿Recuerdas el primer ordinograma del apartado 3? Simplemente calculaba el área de un cuadrado de lado 5 y ni si quiera mostraba el resultado por pantalla...

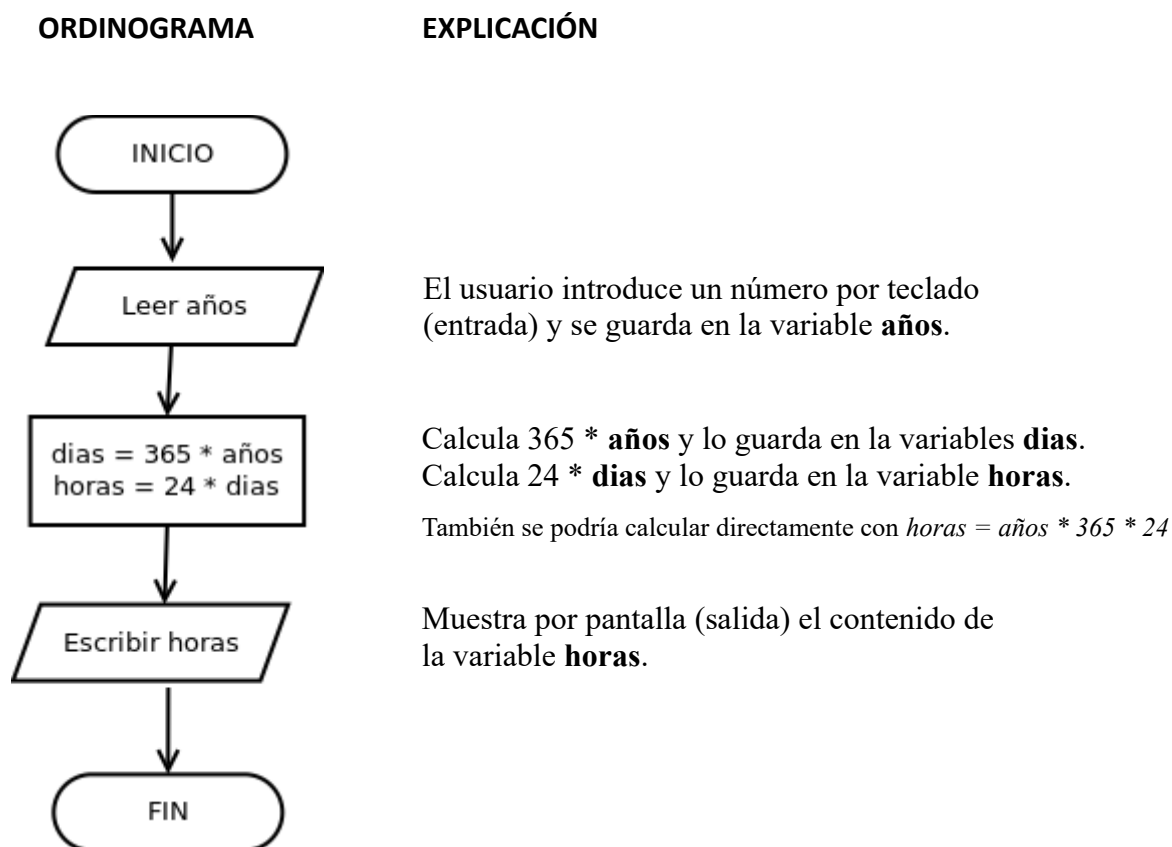
Vamos a modificarlo añadiendo entrada y salida de modo que primero le pida al usuario que introduzca el lado del cuadrado, luego calcule el área, y por último la muestre.



Si creáramos un programa siguiendo este algoritmo mejorado (con entrada y salida) **nos permitiría utilizarlo para calcular el área de cualquier cuadrado**. Eso es mucho más útil y general que el programa original que solo calculaba el área un cuadrado de lado 5.

Por ello, **añadir entrada y salida a los programas es fundamental** para que sean de utilidad.

Ahora vamos a modificar el ordinograma que calculaba los días y horas para añadirle entrada y salida. Queremos que el programa le pida al usuario que introduzca por teclado los años, calcule los días y las horas, y lo muestre por pantalla.



De nuevo, si creáramos un programa que siguiera los pasos de este ordinograma, lo podríamos utilizar para calcular el número de horas de los años que quisiéramos, tantas veces como quisiéramos.



Ahora es un buen momento para hacer los **ejercicios del 1 al 7**.

5. ESTRUCTURAS DE CONTROL

Hasta ahora hemos visto algoritmos (representados como ordinogramas) en los que las instrucciones se ejecutan secuencialmente (una después de la otra). Pero a menudo es necesario diseñar algoritmos cuyas instrucciones no se ejecuten secuencialmente, para lo que es necesario utilizar estructuras de control.

Las estructuras de control son utilizadas para controlar la secuencia (el orden) en el que se ejecutan las instrucciones.

Existen dos tipos:

- **Estructuras alternativas:** Permiten alternar entre distintas instrucciones (ejecutar unas u otras) dependiendo de una condición. Pueden ser simples, dobles o múltiples.
- **Estructuras repetitivas:** Permiten repetir instrucciones (ejecutarlas varias veces).

En esta unidad nos vamos a centrar en las estructuras alternativas.

En la siguiente unidad veremos las estructuras repetitivas.

6. ESTRUCTURAS ALTERNATIVAS

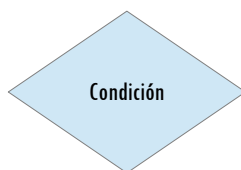
Controlan la ejecución o la no ejecución de una o más instrucciones en función de que se cumpla una condición. Dicho de otra manera, **se utilizan para que sucedan cosas distintas dependiendo de una condición.**

Por ejemplo, al introducir una contraseña si esta es correcta se iniciará sesión, pero si es incorrecta mostrará un mensaje de error. Por poner otro ejemplo, cuando aprietas una letra o un número del teclado ésta se mostrará por pantalla, pero si es la tecla de intro entonces el cursor bajará a la siguiente línea.

Puede parecer obvio pero esto sucede así porque un programador ha utilizado una estructura alternativa para indicar exactamente qué debe suceder en cada caso. **Las estructuras alternativas son muy importantes para establecer distintos comportamientos a un programa.**

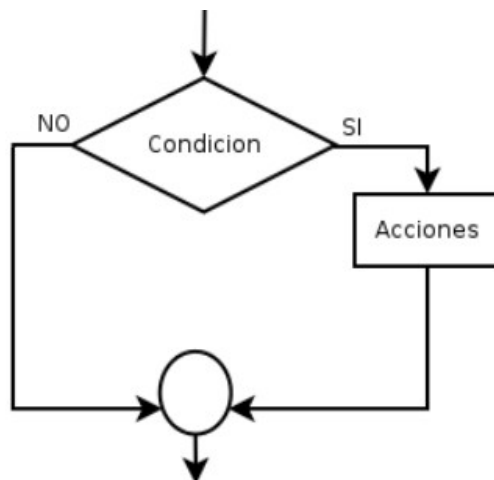
Existen tres tipos de estructuras alternativas: Simple, Doble y Múltiples.

Todas ellas utilizan **condiciones**, como (*precio > 200*), (*edad >= 18*), (*contraseña == "1234"*), etc. En un ordinograma una condición se expresa mediante un **rombo**.

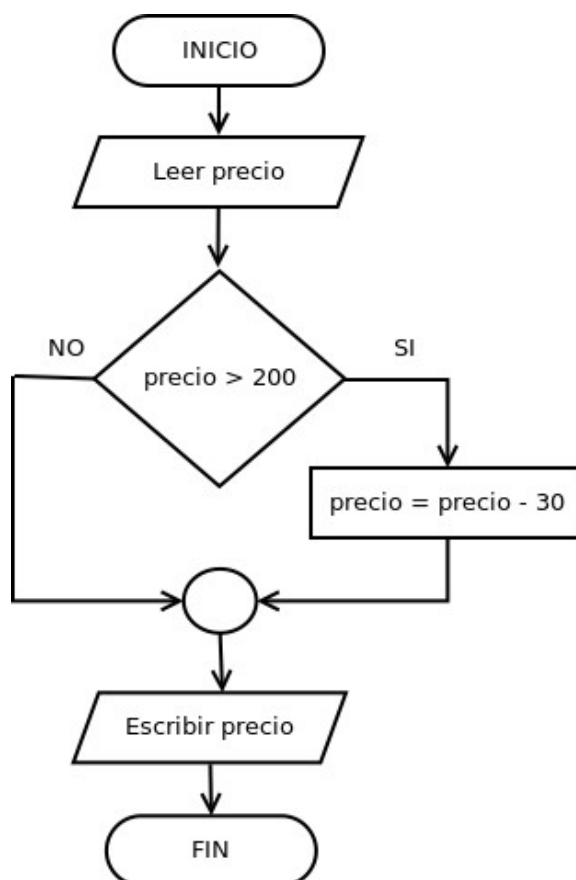


6.1 Estructura alternativa simple

La estructura alternativa simple es muy sencilla. Si la condición es verdadera se ejecutará una o varias instrucciones concretas, pero si es falsa éstas no se ejecutarán. Se representa así:

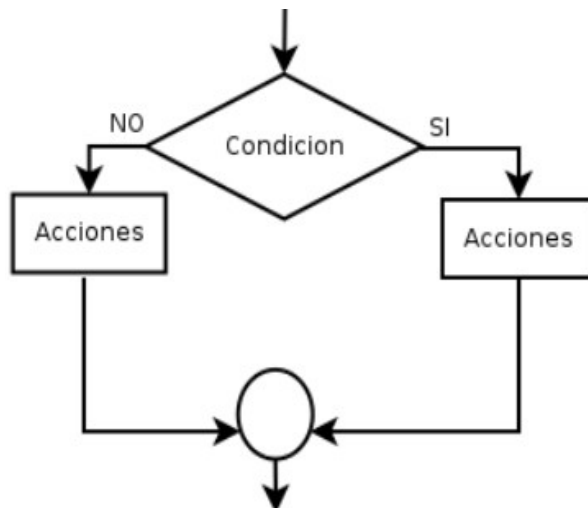


Veamos un ejemplo sencillo. Un programa que lee por teclado el precio inicial de un producto, si es superior a 200 € se le aplica un descuento de 30 €, y por último lo muestra por pantalla.

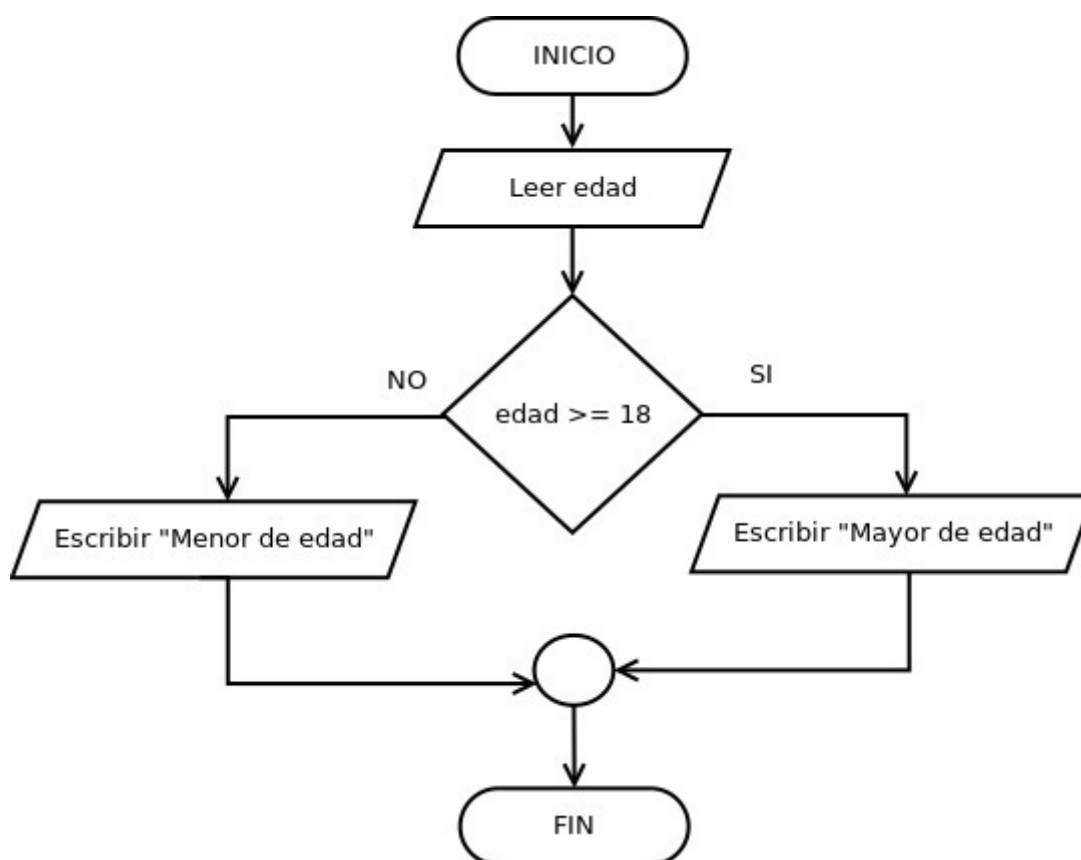


6.2 Estructura alternativa doble

La estructura alternativa doble es muy similar a la simple. La única diferencia es que si la condición es cierta se ejecutarán unas instrucciones y si es falsa se ejecutarán otras distintas.



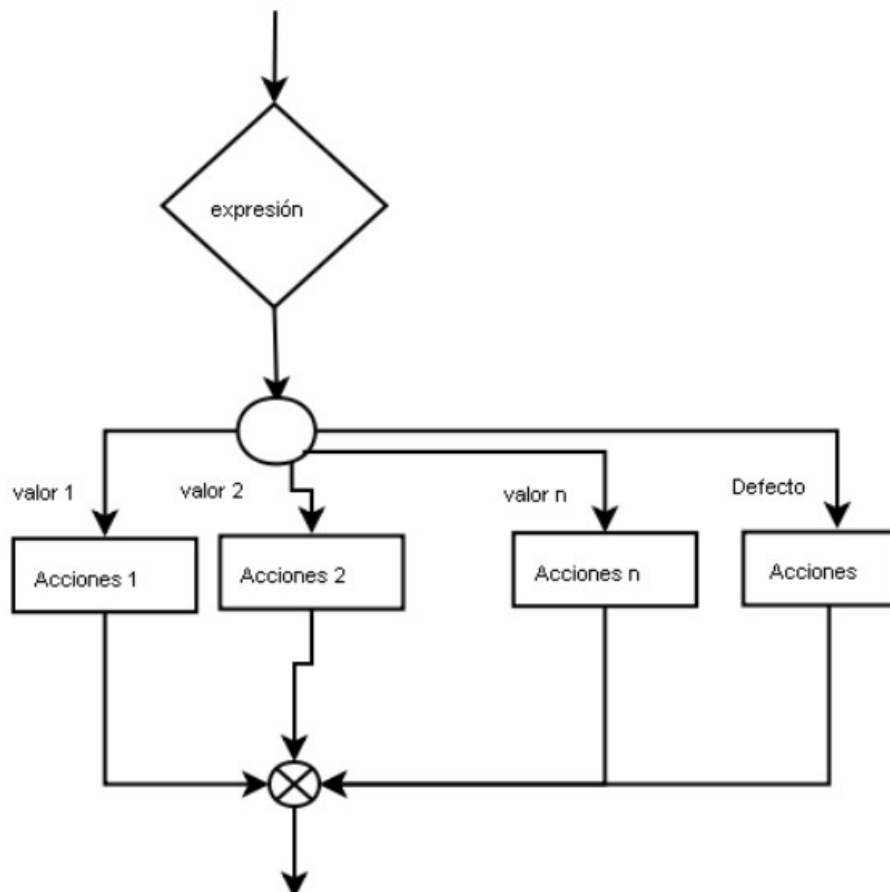
Veamos un ejemplo. Un programa que pide la edad por teclado, si es mayor o igual a 18 mostrará por pantalla el texto "Mayor de edad", en caso contrario mostrará "Menor de edad".



6.3 Estructura de alternativa múltiple

En la estructura alternativa múltiple se pueden especificar distintas instrucciones dependiendo del valor de una expresión. ¡Ojo! de una expresión, no de una condición. La expresión dará un resultado concreto (por ejemplo 10, -5, 0.25, etc.) y se ejecutarán las instrucciones asociadas a cada valor.

Si el resultado de la expresión no es igual a ninguno de los valores indicados, se ejecutarán las instrucciones por defecto (si se ha indicado, esto es opcional).



ESTRUCTURAS REPETITIVAS

7. INTRODUCCIÓN

Las **instrucciones repetitivas (o bucles)** son aquellas que **permiten variar o alterar la secuencia normal de ejecución de un programa** haciendo posible que un grupo de operaciones (acciones) se repita un número determinado o indeterminado de veces, dependiendo del cumplimiento de una condición.

Veremos tres tipos: Bucle Mientras (WHILE), Bucle Hacer-Hasta (DO-WHILE) y Bucle Para (FOR)

8. ESTRUCTURA MIENTRAS (WHILE)

En la estructura Mientras o **“WHILE”** el bloque de acciones **se repite mientras la condición sea cierta**, evaluándose siempre la condición antes de entrar en el bucle, por ello es posible que las acciones no se ejecuten nunca.

Pseudocódigo	Ordinograma
Mientras Condición hacer Instrucción 1 Instrucción 2 ... Instrucción N FinMientras	<pre> graph TD Start(()) --> Condicion{Condicion} Condicion -- Falso --> Exit(()) Condicion -- Verdadero --> Accion1[Accion 1] Accion1 --> Accion2[Accion 2] Accion2 --> Accion3[Accion 3] Accion3 --> AccionN[Accion N] AccionN --> Condicion </pre>

9. ESTRUCTURA HASTA (DO-WHILE)

📖 En la estructura hasta o “**DO-WHILE**”, el bloque de instrucciones **se repite mientras que la condición sea cierta**, y la condición se evalúa al final del bloque por lo que siempre se ejecutarán al menos una vez el bloque de instrucciones.

Pseudocódigo	Ordinograma
<pre>Repetir Instrucción 1 Instrucción 2 ... Instrucción N Mientras Condición.</pre>	<pre>graph TD Start(()) --> A1[Accion 1] A1 --> A2[Accion 2] A2 --> A3[Accion 3] A3 --> AN[Accion N] AN --> Cond{Condicion} Cond -- Verdadero --> A1 Cond -- Falso --> End(())</pre> <p>The flowchart illustrates the Do-While loop structure. It begins with an entry point (downward arrow) leading to a sequence of action boxes: 'Accion 1', 'Accion 2', 'Accion 3', and 'Accion N'. Following these actions is a decision diamond labeled 'Condicion'. If the condition is 'Verdadero' (True), the flow loops back to the entry point before 'Accion 1'. If the condition is 'Falso' (False), the flow exits the loop downwards.</p>

10. ESTRUCTURA PARA (FOR)

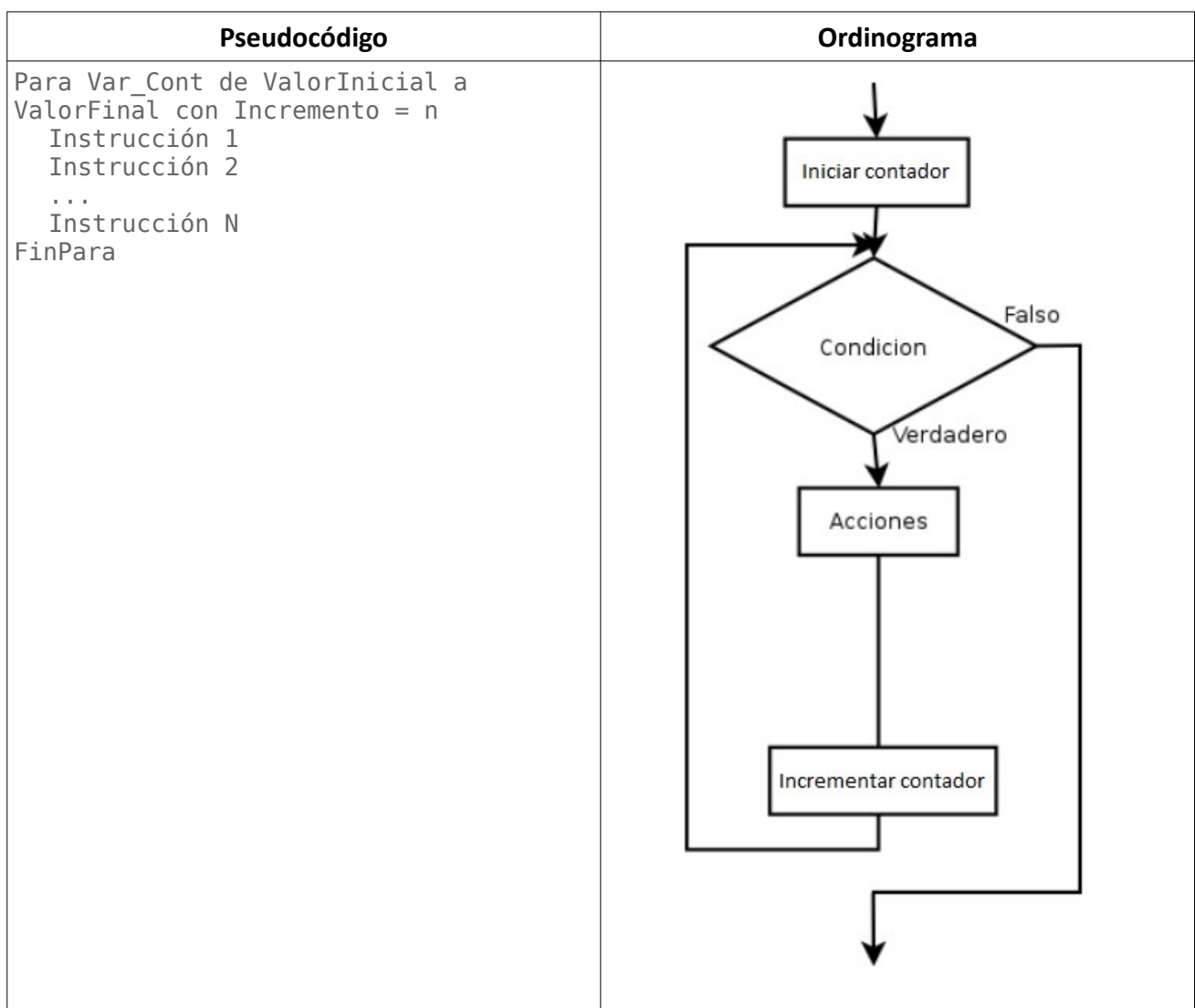
En la estructura Para o “FOR” se conoce de antemano el número de veces que se ejecutará el bloque de instrucciones.



El bloque de acciones **se repite mientras que la condición sea cierta, evaluándose** siempre la condición **antes de entrar en el bucle**, por ello es posible que las acciones no se ejecuten nunca.

Esta explicación es idéntica a la del bucle WHILE, pero un bucle FOR debe cumplir las siguientes características:

1. La variable contador se inicializa con un valor inicial.
2. La condición siempre debe ser: $\text{variable_contador} \leq \text{valor_final}$
3. En cada interacción, la variable contador se incrementa en un determinado valor.



11. FORMAS DE ACABAR UN BUCLE

Uno de los peligros que se corren cuando se escribe un bucle es que éste no acabe nunca, lo que se denomina **bucle infinito**, para no cometer este error grave debemos recordar que las condiciones de los bucles deben poder cambiar dentro del bucle, es decir que si por ejemplo utilizamos una variable comparada con una constante, dicha variable debe poder cambiar de valor dentro del bucle.

⚡ Las estructuras repetitivas deben incluir un mecanismo para que éstas se acaben.

Algunos de los métodos más usados para esa labor son los siguientes:

1. Cuando sabemos el número de veces que se va a repetir la estructura, utilizaremos un contador.

Por ejemplo, en un enunciado del tipo: “imprimir la tabla del 7”, sabemos que el proceso va desde 1 a 10, por lo tanto, usaremos un contador.

```
cont = 1
Mientras cont <=10
    Escribir cont * 7
    cont = cont + 1
FinMientras
```

2. Preguntando si queremos seguir en el bucle.

Por ejemplo, en un ejercicio del tipo “introducir N alumnos y hallar su media”, debemos preguntar si queremos introducir más alumnos:

```
. . .
seguir="s"
Mientras ((seguir="s") o (seguir="S"))
    . . .
    Escribir "Introducir más alumnos?"
    Leer seguir
FinMientras
. . .
```

3. Usando un valor centinela.

Así, en el caso del siguiente problema: “Introducir N notas hasta introducir un 10”:

```
. . .
Leer nota
Mientras (nota <> 10)
    . . .
    Leer nota
FinMientras
. . .
```


4. Usando un interruptor que tomará el valor lógico true o false.

En un ejemplo como “Repetir ciertas instrucciones mientras la condición sea cierta”:

```
· · ·  
Mientras (SW = Verdadero)  
· · ·  
FinMientras  
· · ·
```

12. ELEMENTOS AUXILIARES

Los **elementos auxiliares** son **variables que realizan funciones específicas dentro de un programa**, y por su gran utilidad, frecuencia de uso y peculiaridades, conviene hacer un estudio separado de las mismas.

12.1 Contadores

Si vamos a repetir una acción un número determinado de veces y esa variable se va a incrementar siempre en una cantidad constante, se denomina **contador**. Sería útil llamarla algo así como CONT, CONTA, CONTADOR... Ai tuviéramos varios contadores dentro de un programa podríamos llamarlos CONT1, CONT2...

Se utilizan en los siguientes casos:

- Para contabilizar el número de veces que es necesario repetir una acción (variable de control de un bucle).
- Para contar un suceso particular solicitado por el enunciado del problema. Un contador debe inicializarse a un valor inicial (normalmente a cero) e incrementarse cada vez que ocurra un suceso.

12.2 Acumuladores

Si por el contrario, dicho objeto se va **incrementando de forma variable** se denomina **acumulador**. Deberemos llamarla ACU, ACUM, ACUMULA, ACUMULADOR, SUMA, ... u otra palabra significativa.

Se utiliza en aquellos casos en que se desea obtener el total acumulado de un conjunto de cantidades, siendo inicializado con un valor cero.

También en ocasiones hay que obtener el total acumulado como producto de distintas cantidades, en este caso se inicializará a uno.

Por ejemplo: imprimir la suma de N edades.

12.3 Interruptores

Por último, tenemos ciertas variables que pueden **tomar dos valores: cierto o falso**. Se les denomina **interruptores o switches** y su función es que ciertas instrucciones se ejecuten mientras tenga un valor determinado. A las variables de este tipo las denominaremos SW.

Se utiliza para:

- Recordar que un determinado suceso a ocurrido o no en un punto determinado del programa, y poder así realizar las decisiones oportunas.
- Hacer que dos acciones diferentes se ejecuten alternativamente dentro de un bucle.

Por ejemplo: introducir N edades y acabar al introducir un 99.

13. LICENCIA



[CC BY-NC-SA 3.0 ES](https://creativecommons.org/licenses/by-nc-sa/3.0/es/) Reconocimiento – No Comercial – Compartir Igual (by-nc-sa)

No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original. Esta es una obra derivada de la obra original de Carlos Cacho y Raquel Torres.

13.1 Agradecimientos

Apuntes actualizados y adaptados al CEEDCV a partir de la siguiente documentación:

- [1] Apuntes Programación de José Antonio Díaz-Alejo. IES Camp de Morvedre.