

UF1. PROGRAMACIÓ ESTRUCTURADA

NF2.- PROGRAMACIÓ ESTRUCTURADA

2.1. Estructura de un programa

Para que los programas cumplan los requisitos de legibilidad, fiabilidad, portabilidad, modificabilidad y eficiencia, se utiliza un conjunto de métodos y técnicas que ayudan al desarrollo de programas.

Los métodos de programación que seguiremos durante este curso son el método de programación modular y el de programación estructurada.

Programación modular

Se basa en el diseño descendente (Top-down) mediante descomposiciones sucesivas de un problema complejo en acciones más simples. De esta forma un programa quedará formado por una serie de **módulos**, cada uno de los cuales realiza una parte concreta de la tarea total.

Programación estructurada

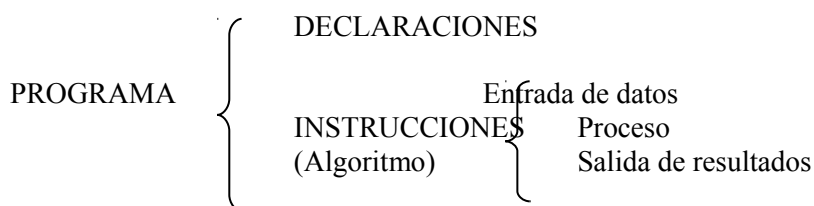
Se basa en el uso exclusivo de las estructuras **secuenciales**, **alternativas (condicionales)** y **repetitivas (iterativas)** para el control del flujo de ejecución de las instrucciones.

2.1.1. Partes de un programa

Todo programa, en general, contiene dos bloques bien diferenciados :

Bloque de declaraciones. En él se especifican todos los objetos que utiliza el programa (constantes, variables, tablas, registros, archivos, etc.) indicando sus características. Este bloque se encuentra localizado siempre antes del bloque de instrucciones.

Bloque de instrucciones (algoritmo). Constituido por el conjunto de acciones que se han de realizar para la obtención de los resultados deseados. Dentro de este bloque podemos diferenciar tres partes fundamentales: entrada de datos, proceso, salida de datos o resultados.



2.2. Algoritmos

Un algoritmo se puede definir como la descripción abstracta de todas las acciones u operaciones que debe realizar un ordenador de forma clara y detallada para resolver un problema. La diferencia con la definición de programa está en el nivel de abstracción, ya que el programa es un conjunto de instrucciones en un lenguaje de programación concreto.

Según la RAE: conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.

El lenguaje algorítmico es aquel que implementa una solución teórica a un problema indicando las operaciones a realizar y el orden en el que se deben efectuarse. Por ejemplo en el caso de que nos encontremos en casa con una bombilla fundida en una lámpara, un posible algoritmo sería:

- (1) Comprobar si hay bombillas de repuesto
- (2) En el caso de que las haya, sustituir la bombilla anterior por la nueva
- (3) Si no hay bombillas de repuesto, bajar a comprar una nueva a la tienda y sustituir la vieja por la nueva

Los algoritmos son la base de la programación de ordenadores, ya que los programas de ordenador se puede entender que son algoritmos escritos en un código especial entendible por un ordenador. Lo malo del diseño de algoritmos está en que no podemos escribir lo que deseamos, el lenguaje a utilizar no debe dejar posibilidad de duda, debe recoger todas las posibilidades. Por lo que los tres pasos anteriores pueden ser mucho más largos:

[1] Comprobar si hay bombillas de repuesto

(1.1) Abrir el cajón de las bombillas

(1.2) Observar si hay bombillas

[2] Si hay bombillas:

(2.1) Coger la bombilla

(2.2) Coger una silla

(2.3) Subirse a la silla

(2.4) Poner la bombilla en la lámpara

[3] Si no hay bombillas

(3.1) Abrir la puerta

(3.2) Bajar las escaleras....

Como se observa en un algoritmo las instrucciones pueden ser más largas de lo que parecen, por lo que hay que determinar qué instrucciones se pueden utilizar y qué instrucciones no se pueden utilizar. En el caso de los algoritmos preparados para el ordenador, se pueden utilizar sólo instrucciones muy concretas.

2.2.1. Características de los algoritmos

Características que deben de cumplir los algoritmos obligatoriamente

- **Un algoritmo debe resolver el problema para el que fue formulado.**
Lógicamente no sirve un algoritmo que no resuelve ese problema. En el caso de los programadores, a veces crean algoritmos que resuelven problemas diferentes al planteado.
- **Los algoritmos son independientes del ordenador.**
Los algoritmos se escriben para poder ser utilizados en cualquier máquina.
- **Los algoritmos deben de ser precisos.**
Los resultados de los cálculos deben de ser exactos, de manera rigurosa. No es válido un algoritmo que sólo aproxime la solución.

- **Los algoritmos deben de ser finitos.**
Deben de finalizar en algún momento. No es un algoritmo válido aquel que produce situaciones en las que el algoritmo no termina.
- **Los algoritmos deben de poder repetirse.**
Deben de permitir su ejecución las veces que haga falta. No son válidos los que tras ejecutarse una vez ya no pueden volver a hacerlo por la razón que sea.

Características aconsejables para los algoritmos

- **Validez.**
Un algoritmo es válido si carece de errores. Un algoritmo puede resolver el problema para el que se planteó y sin embargo no ser válido debido a que posee errores
- **Eficiencia.**
Un algoritmo es eficiente si obtiene la solución al problema en poco tiempo. No lo es si es lento en obtener el resultado.
- **Óptimo.**
Un algoritmo es óptimo si es el más eficiente posible y no contiene errores. La búsqueda de este algoritmo es el objetivo prioritario del programador. No siempre podemos garantizar que el algoritmo hallado es el óptimo, a veces sí.

2.2.2. Elementos que conforman un algoritmo

- ◆ **Entrada.** Los datos iniciales que posee el algoritmo antes de ejecutarse.
- ◆ **Proceso.** Acciones que lleva a cabo el algoritmo.
- ◆ **Salida.** Datos que obtiene finalmente el algoritmo.

Entrada de datos

La constituyen todas las instrucciones que toman los datos de entrada desde un dispositivo externo y los almacenan en la memoria central para que puedan ser procesados. También se consideran dentro de esta parte las instrucciones que se encargan de comprobar la corrección de los datos.

Proceso

Está formado por las instrucciones que modifican los datos de entrada para conseguir los resultados o datos de salida y depositarlos de nuevo en memoria central.

Salida de datos o resultados

Conjunto de instrucciones que toman los datos finales (resultados) de la memoria central y los envían a los dispositivos externos.

Para el diseño de algoritmos se utilizan dos tipos de técnicas: diagramas de flujo (ordinogramas) y Pseudocódigo.

2.3. Diagramas de flujo y pseudocódigo

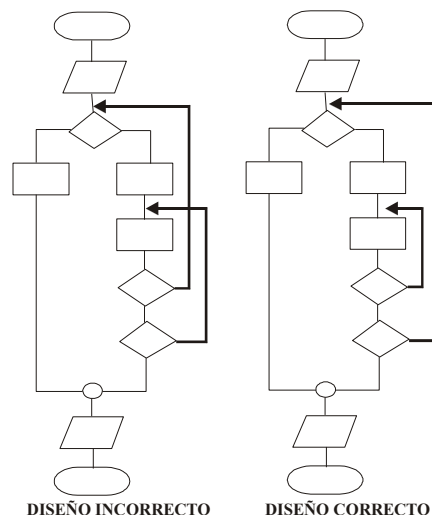
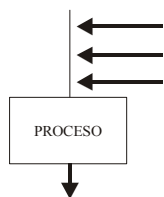
Diagramas de flujo

Se definen como la representación gráfica que mediante el uso de símbolos estándar conectados o unidos mediante líneas de flujo, muestran la secuencia lógica de las operaciones o acciones que constituyen el algoritmo de resolución del problema para el que ha sido creado. Si los algoritmos son complejos, este tipo de esquemas no son adecuados.

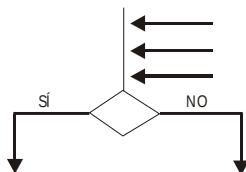
- El diseño de todo diagrama de flujo debe reflejar:
 - Un inicio que marca el comienzo de ejecución del programa y que viene determinado por la palabra “INICIO”.
 - La secuencia de operaciones, lo más detallada posible y siguiendo siempre el orden en el que se deberán ejecutar (de arriba abajo y de izquierda a derecha).
 - Un fin que marca la finalización de ejecución del programa y que viene determinado por la palabra “FIN”.

Las reglas que hay que seguir para la confección de un diagrama de flujo son las siguientes:

- Todos los símbolos utilizados en el diseño deben estar conectados por medio de líneas de conexión.
- Está prohibido el cruce de líneas de conexión, pues ello nos indica que el ordinograma no está correctamente diseñado.
- A un símbolo de proceso pueden llegarle varias líneas de conexión, pero de él sólo puede salir una.

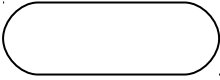
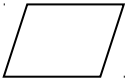
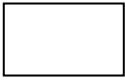
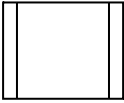

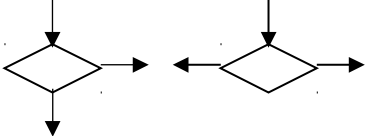
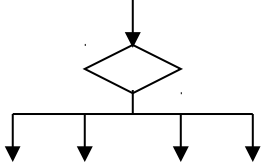

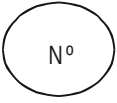




- A un símbolo de decisión pueden llegarle varias líneas de conexión, pero de él sólo puede salir una línea de entre las dos posibilidades existentes (verdadero o falso).



- A un símbolo de inicio de proceso no llega ninguna línea de conexión y de él sólo puede partir una línea de conexión.
- A un símbolo de final de proceso pueden llegar varias líneas de conexión, pero de él no puede partir ninguna.

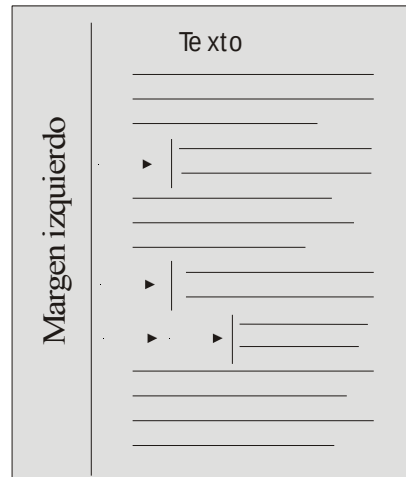
Símbolos de operación

Símbolo	Función
	Terminal (marca el inicio, final o una parada necesaria realizada en la ejecución del programa)
	Operación de E/S en general
	Proceso u operación en general
	Subprograma o subrutina
	Modificación de instrucciones o inicializaciones
	Decisión de dos salidas
	Decisión múltiple
	Conector (se utiliza para el reagrupamiento de líneas de flujo)
	Conector de líneas de flujo en la misma página (utilizado para enlazar dos partes cualesquiera del diseño)
	Conector de líneas de flujo en distintas páginas)
	Comentario

Pseudocódigo

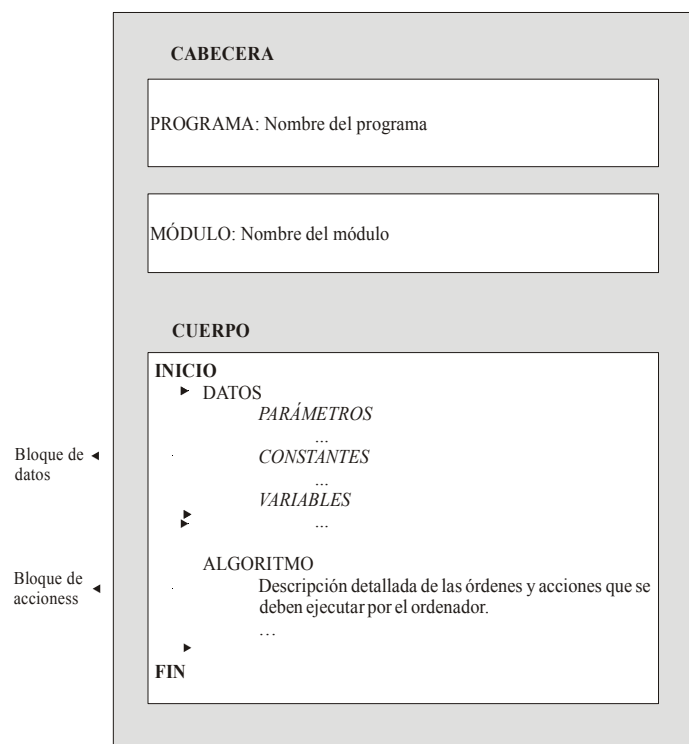
El pseudocódigo es un lenguaje intermedio entre el lenguaje natural y el lenguaje de programación. Mediante el pseudocódigo podemos describir la solución de un problema en forma de algoritmo, utilizando palabras y frases del lenguaje natural sujetas a unas determinadas y estrictas reglas.

La escritura de un algoritmo mediante el uso de esta herramienta exige la “identación” o “sangría” del texto en el margen izquierdo de las diferentes líneas.



Sangría o indentación del texto

Estructura general de un algoritmo en notación pseudocodificada:



En pseudocódigo los comentarios que se deseen poner (y esto es una práctica muy aconsejable) se ponen con los símbolos // al principio de la línea de comentario (en algunas notaciones se escribe **). Cada línea de comentario debe comenzar con esos símbolos:

```
inicio
instrucciones
//comentario
//comentario
instrucciones
fin
```


2.4. Tipos de instrucciones

Una instrucción es una acción que genera cambios previstos en la ejecución de un programa.

- ◆ **Primitivas.** Son acciones sobre los datos del programa. Son:
 - Asignación
 - Instrucciones de Entrada/Salida
- ◆ **Declaraciones.** Obligatorias en el pseudocódigo, opcionales en otros esquemas. Sirven para advertir y documentar el uso de variables y subprogramas en el algoritmo.
- ◆ **Control.** Sirven para alterar el orden de ejecución del algoritmo. En general el algoritmo se ejecuta secuencialmente. Gracias a estas instrucciones el flujo del algoritmo depende de ciertas condiciones que nosotros mismos indicamos.

Instrucciones de declaración de datos

Son las utilizadas para informar al procesador del espacio que debe reservar en la memoria para albergar un dato. Para ello se debe indicar el identificador del dato (nombre) y el tipo. Es aconsejable al escribir pseudocódigo indicar las variables que se van a utilizar (e incluso con un comentario indicar para qué se van a usar). En el caso de los diagramas de flujo no se utilizan lo que fomenta malos hábitos.

Esto se hace mediante la sección del pseudocódigo llamada var, en esta sección se colocan las variables que se van a utilizar. Esta sección se coloca antes del inicio del algoritmo. Y se utiliza de esta forma:

```
programa nombreDePrograma
var
    identificador1: tipoDeDatos //comentario
    identificador2: tipoDeDatos //comentario
    ....
inicio
    instrucciones
fin
```

Ejemplo de declaración:

```
var
    numero_cliente: entero // código único de cada cliente
    valor_compra: real //lo que ha comprado el cliente
    descuento: real //valor de descuento aplicable al cliente
```

También se pueden declarar de esta forma:

```
var
    numero_cliente: entero // código único de cada cliente
    valor_compra, //lo que ha comprado el cliente
    descuento :real //valor de descuento aplicable al cliente
```

La coma tras valor_compra permite declarar otra variable real.

Declaración de constantes:

```
programa ejemplo1
const
    PI=3.141592
    NOMBRE="Jose"
var
    edad: entero
    sueldo: real
inicio
    ....
```

A las constantes se las asigna un valor mediante el símbolo =. Ese valor permanece constante (pi siempre vale 3.141592). Es conveniente (aunque en absoluto obligatorio) utilizar letras mayúsculas para declarar variables.

Instrucciones primitivas (acciones simples)

Son aquellas que el procesador ejecuta de forma inmediata. Son la asignación, la entrada y la salida.

Asignación. Permite almacenar un valor en una variable. Para asignar el valor se escribe el símbolo \leftarrow , de modo que:

identificador \leftarrow valor

El identificador toma el valor indicado.

$x \leftarrow 8$

Ahora x vale 8. Se puede utilizar otra una expresión en lugar de un valor.

$y \leftarrow 9$

$x \leftarrow y$

x vale ahora lo que vale y , es decir x vale 9.

En las instrucciones de asignación se pueden utilizar expresiones más complejas con ayuda de los operadores.

Ejemplo:

$x \leftarrow (y * 3) / 2$

Es decir x vale el resultado de multiplicar el valor de y por tres y dividirlo entre dos.

Entrada. Toma un dato del dispositivo de entrada (teclado) y lo almacena en una variable. Si se leen varias variables se pueden colocar éstas en una misma instrucción separándolas por comas. Se hace mediante la orden **leer** en la que entre paréntesis se indica el identificador de la variable que almacenará lo que se lea. Ejemplo (pseudocódigo):

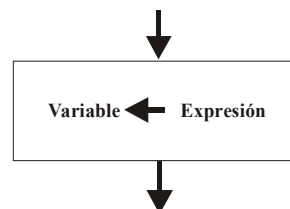
leer (x)

x contendrá el valor leído desde el teclado. Se pueden leer varias variables a la vez:

leer (x, y, z)

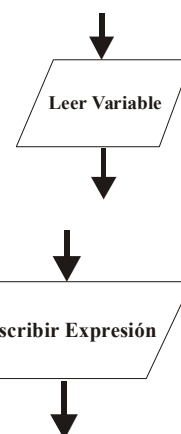
Salida. Funciona como la anterior pero usando la palabra **escribir**. Simula la salida de datos del algoritmo por pantalla.

escribir (x, y, z)



Ejemplo:

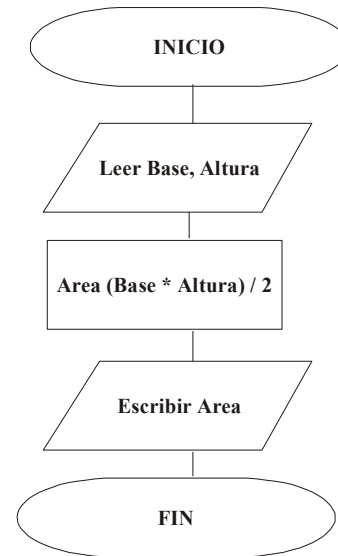
variable o
Ejemplo:



EJEMPLO: Algoritmo que calcula el área de un triángulo utilizando como métodos de

representación el diagrama de flujo y el pseudocódigo.

```
PROGRAMA: Area_del_triángulo  
var  
    area, base, altura: real  
    Base    Numérico Real  
inicio  
    leer(base, altura)  
    area ← (base * altura) / 2  
    escribir(area)  
fin
```



EJERCICIOS:

1) Programa que toma como dato de entrada un número que corresponde a la longitud de un radio y nos calcula y escribe la longitud de la circunferencia, el área del círculo y el volumen de la esfera que se corresponden con dicho radio.

$$l = 2\pi r \quad a = \pi r^2 \quad v = \frac{4}{3} \pi r^3$$

2) Programa que lee una temperatura en la escala centígrada y nos calcula y escribe su valor en la escala Fahrenheit. $^{\circ}\text{C} / 100 = (^{\circ}\text{F} - 32) / 180$

*****USO DFD.EXE CON EJEMPLOS DE LOS EJERCICIOS 1 Y 2*****

2.5. Estructuras de control (sentencias de control)

Son la base de la programación estructurada. Son utilizadas para controlar la secuencia de ejecución de un programa y, en consecuencia, determinados bloques de instrucciones.

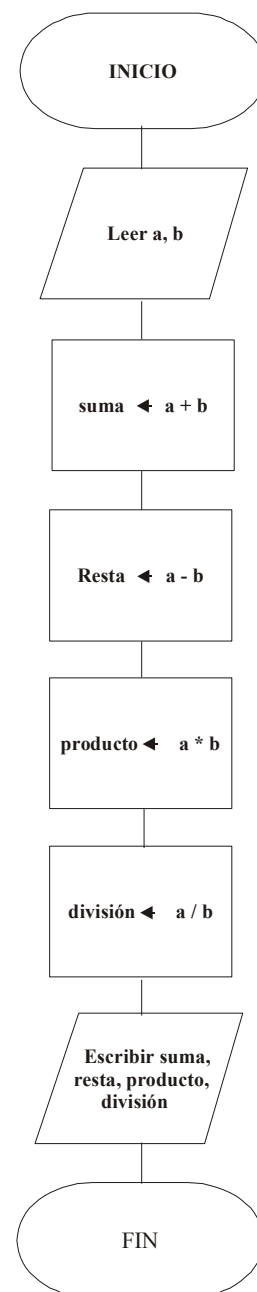
Estructura secuencial

El orden de ejecución de las instrucciones de un programa es secuencial, es decir, que las instrucciones se ejecutan de arriba abajo y de izquierda a derecha una detrás de otra.

Instrucción 1
Instrucción 2
Instrucción 3
...
Instrucción n

EJEMPLO: Algoritmo que lee dos valores reales y calcula primero la suma, después la resta, a continuación el producto y seguidamente la división de ambos valores, escribiendo finalmente el resultado obtenido en cada una de estas operaciones.

```
PROGRAMA Oper_aritméticas
var
  a, b, suma, resta, producto, división: real
inicio
  leer (a, b)
  suma ← a + b
  resta ← a - b
  producto ← a * b
  división ← a / b
  escribir (suma, resta, producto, división)
fin
FPROGRAMA
```



2.5.1. Estructura alternativa

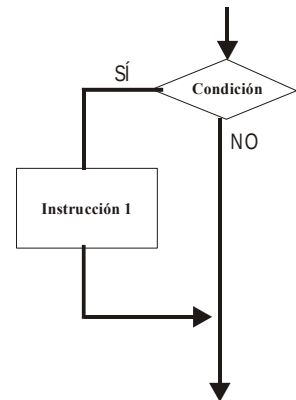
Controla la ejecución o la no ejecución de una o más instrucciones en función de que se cumpla o no una condición previamente establecida.

- **ALTERNATIVA SIMPLE**

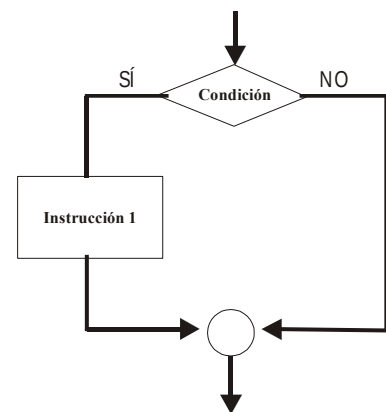
La instrucción 1 se ejecuta si la condición se evalúa a cierto.
En caso contrario no se hace nada.

Puede ser una única instrucción o varias las que se ejecutan.

Hay dos posibles representaciones en el ordinograma.

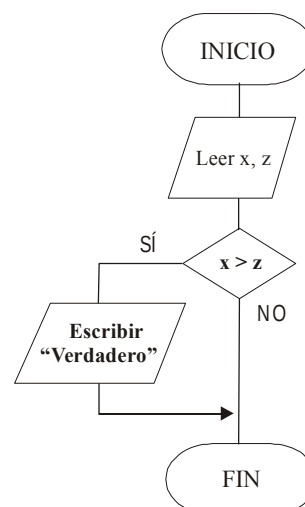


```
si expresión_logica entonces
    instrucciones
fin_si
```



EJEMPLO. Algoritmo que lee dos valores numéricos y los almacena en dos variables de nombre 'x' y 'z', mostrando en aquellos casos en los que 'x' es mayor que 'z' un mensaje que diga "verdadero".

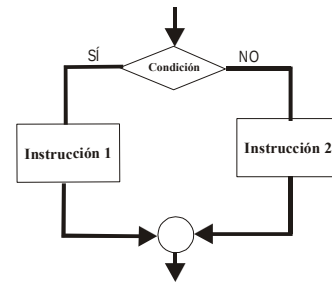
```
PROGRAMA Condición_simple
var
    x, z: entero
inicio
    leer(x, z)
    si x > z entonces
        escribir "Verdadero"
    fin_si
fin
```



- **ALTERNATIVA DOBLE**

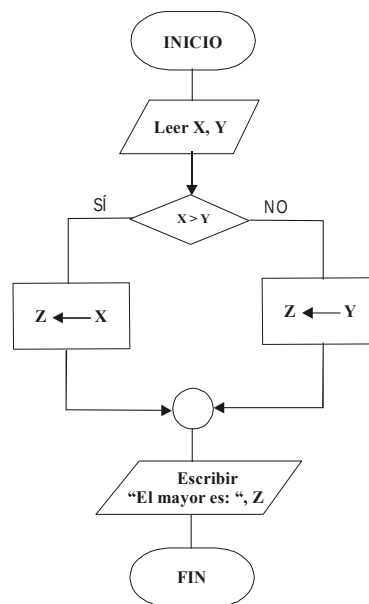
Si la condición se evalúa a cierto se ejecuta la instrucción 1. En caso contrario se ejecuta la instrucción 2.

Puede ser una única instrucción o varias las que se ejecutan o se dejan de ejecutar.



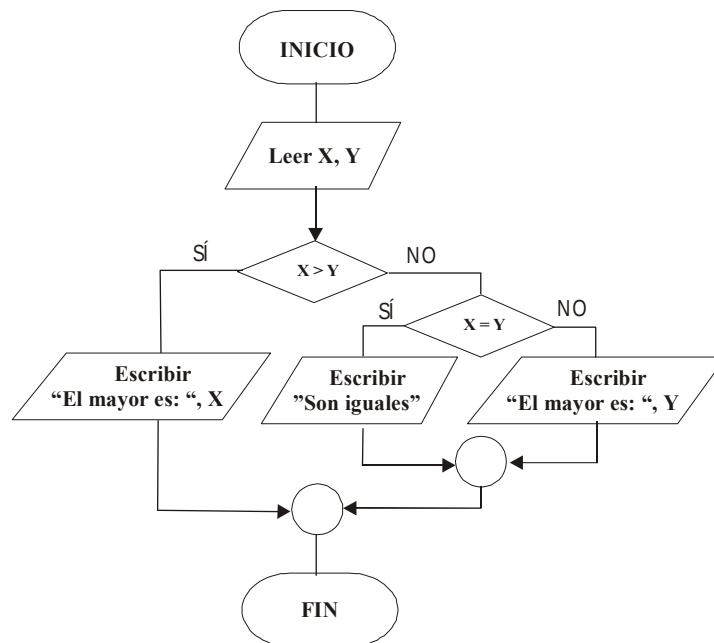
```
si expresión_lógica entonces
    instrucciones //se ejecutan si la expresión
                  es verdadera
sino
    instrucciones //se ejecutan si la expresión
                  es falsa
fin_si
```

EJEMPLO. Algoritmo que lee dos valores numéricos distintos 'x' e 'y' y determina cuál es el mayor dejando el resultado en una tercera variable de nombre 'z'.



EJEMPLO. Algoritmo que lee dos valores numéricos 'x' e 'y', determina si son iguales y en caso de no serlo, indica cuál de ellos es el mayor.

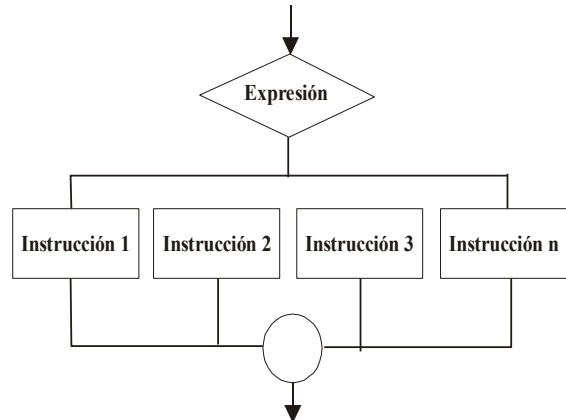
```
PROGRAMA Condición_anidada
var
    x, y: real
inicio
    leer (x, y)
    si x > y entonces
        escribir ("El mayor es : ", x)
    sino
        si x = y entonces
            escribir ("son iguales")
        sino
            escribir ("El mayor es: ", y)
        fin_si
    fin_si
fin
```



- **ALTERNATIVA MÚLTIPLE**

Una expresión alfanumérica o numérica entera puede tomar diversos valores, según cual sea su valor se ejecutará una instrucción determinada.

opción Expresión de	
v1:	Instrucción 1
v2:	Instrucción 2
...	
vn:	Instrucción n
otro:	Instrucción x
fin_opción	



EJEMPLO. Algoritmo que lee una variable de tipo carácter llamada EstadoCivil. Si la letra es una 's' escribe "SOLTERO", si es una 'c' escribe "CASADO", si es una 'v' escribe "VIUDO", si es una 'd' escribe "DIVORCIADO", y si no es ninguna de las anteriores escribe "ERROR".

```
PROGRAMA Estado_Civil
var
    EstadoCivil: carácter
inicio
    leer(EstadoCivil)
    opción EstadoCivil de
        's':    escribir("SOLTERO")
        'c':    escribir("CASADO")
        'v':    escribir("VIUDO")
        'd':    escribir("DIVORCIADO")
        Otro:   escribir("ERROR")
    fin_opción
fin
```


EJEMPLO:

```
programa pruebaSelMultiple
var
    x: entero
inicio
    escribir("Escribe un número del 1 al 4 y te diré si es par o impar")
    leer(x)
    opcion x de
        1: escribir("impar")
        2: escribir("par")
        3: escribir("impar")
        4: escribir("par")
        otro: escribir("error eso no es un número de 1 a 4")
    fin_opcion
fin
```

Se puede escribir también:

```
opcion x de
    1,3: escribir("impar")
    2,4: escribir("par")
    Otro: escribir("error eso no es un número de 1 a 4")
fin_opcion
```

Es decir el valor en realidad puede ser una lista de valores. Para indicar esa lista se pueden utilizar expresiones como:

1..3 De uno a tres (1,2 o 3)

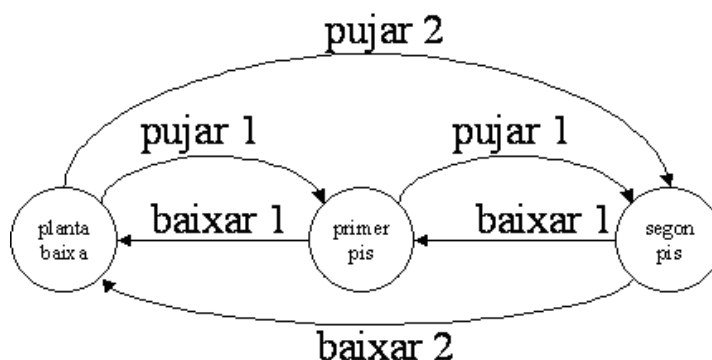
>4 Mayor que 4

>5 Y <8 Mayor que 5 y menor que 8

7,9,11,12 7,9,11 y 12. Sólo esos valores (no el 10 o el 8 por ejemplo)

Sin embargo estas últimas expresiones no son válidas en todos los lenguajes (por ejemplo el C no las admite).

- 1.Desenvolpeu un programa que demani dos números i escrigui els dos números ordenats creixentment (de menor a major). Ex. si els números són 3 i 2, el resultat serà 2 3
- 2.Desenvolpeu un programa que demani un mes i un any, i escrigui el mes anterior i el mes següent. Ex. Si l'usuari introdueix mes:10 i any 2003, el resultat serà anterior:9/2003 i posterior 11/2003
- 3.Desenvolpeu un programa que demani un any i escrigui si és de traspàs o no. Sabem que són de traspàs tots els anys múltiples de 4, excepte els que siguin múltiples de 100 sense que siguin múltiples de 400.
- 4Desenvolpeu un programa que demani tres números i escrigui els tres números ordenats creixentment (de menor a major). Ex. si els números són 3, 4 i 2, el resultat serà 2 3 4.
- 5Desenvolpeu un programa que demani una hora expressada en hores, minuts i segons i mostri la hora, minuts i segons que serà transcorregut un segon.
- 6.Desenvolpeu una calculadora senzilla que demani a l'usuari un primer operand numèric, una operació entre (+, -, *, /) i un segon operand, i escrigui el resultat d'aplicar l'operació als operands.
- 7.En un determinat comerç es realitza un descompte depenent del preu de cada producte. Si el preu es inferior a 10 €, no es fa descompte; si és major o igual a 10 € i menor que 100 €, es fa un 5 per 100 de descompte, i si és major o igual a 100 €, es fa un 10 per 100 de descompte. Realitzar el programa que llegeix el preu d'un producte i ens calcula i escriu el preu final.
- 8.Desenvolpeu un programa que demani a l'usuari que introdueixi un preu en € i la quantitat de € que paga. El programa comparará les dues quantitats i escriurà els € que li falten per pagar o bé els que li han de tornar. Ex. Si l'usuari introdueix preu=102€ i paga=150€, el programa li dirà "Sobren 48€". Si l'usuari introdueix preu=102€ i paga=100€, el programa li dirà "Falten 2€"
- 9.Desenvolpeu un metge virtual "especialista" en refredats. El nostre metge, només sap que:
 - a.si el malalt presenta esternuts i mal de cap llavors, si té problemes d'estómac li recomanarà prendre paracetamol i si no en té li proposarà prendre àcid acetil salicilic (AAS).
 - b.si el malalt ens diu que té tos llavors, si és massa jove (menor de 12 anys) li recomanarà un caramel de mel i altrament li proposarà un caramel d'eucaliptus
 - c.si no presenta cap dels anteriors símptomes, el metge proposarà al pacient que vingui a la seva consulta presencial per poder examinar-lo
- 10.Desenvolpem un ajudant infantil per decidir que fer davant un semàfor. El programa demanarà de quin color està el semàfor i segons la resposta recomanarà passar, esperar, o córrer
- 11.Considerem un ascensor d'un edifici amb planta baixa i dos pisos que tingui els següents botons: 'pujar 1', 'pujar 2', 'baixar 1' i 'baixar 2'. L'ascensor es comporta, a partir dels botons esmentats, segons el següent diagrama de transició d'estats:



Es demana desenvolupar un programa que simuli el funcionament d'aquest ascensor. El programa demanarà el pis en que es troba i el botó que es prem, i mostrarà el nou pis.

12. Considerem el mateix ascensor que a l'exercici anterior. Ara afegim que, en cas que el botó seleccionat a una posició determinada de l'ascensor no correspongui amb algun dels representats amb el diagrama d'estats, l'ascensor emet un soroll i es queda al lloc en què estava.

Modifiquem el nostre programa per a afegir la simulació d'aquest comportament. Així, quan el botó que ens indiquin no sigui correcte, escriurem un missatge d'error en comptes del nou pis.