TEMA 2: ALGORITMOS Y PROGRAMAS

Introducción

Un algoritmo es una sucesión finita de pasos no ambiguos, que se ejecutan en un tiempo finito, que le dicen al ordenador lo que hay que hacer en cada momento para llegar a la resolución del problema.

Resaltar que:

... es una cantidad **finita** de pasos que se llevan a cabo en un **tiempo finito** (los bucles infinitos no son algoritmos.)

... es una sucesión de pasos **no ambiguos**, es decir cada paso especifica una tarea determinada a ser realizada en cada momento por el ordenador.

Los pasos a seguir en la resolución de cualquier problema mediante un ordenador son los siguiente:

1. Análisis del problema.
2. Búsqueda del algoritmo.
3. Programación de algoritmo.
4. Traducción y comprobación del programa.

Análisis del problema

a.- Acotar y especificar el problema con total precisión (obtener el máximo de información acerca de lo que debemos resolver y las soluciones a determinar.)

b.- Definir los datos iniciales o de partida (que datos necesitamos proporcionar al problema para resolverlo.)

c.- Definir que datos o resultados debe proporcionar el algoritmo.

Búsqueda del algoritmo

Búsqueda de una sucesión finita de pasos no ambiguos que nos lleven a la resolución del problema:

a.- Selección del mejor algoritmo.

b.- Mejora del algoritmo.

La selección y la mejora se hacen, habitualmente, en función del tiempo de ejecución (aproximadamente la cantidad de instrucciones que se tienen que ejecutar para resolver correctamente el problema.)

Ejemplo 1: Búsqueda de 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.

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.

Ejemplo 2: Suma de una sucesión aritmética

1.- *Análisis del problema.* Descripción del problema: $a_1 + a_2 + a_3 + ... + a_n$ donde $a_2 = a_1 + d$ y en general $a_n = a_1 + (n-1)*d$

Entrada: Primer término (a_1)
Distancia (d)
Número de términos (n)

Salida: Suma de todos los términos $\sum_{i=1}^{n} a_i = S_n$

2.- *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*** 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)

Método 2: ***Aplicación de la fórmula***

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

.2. Mostrar el resultado (S_n)

Para la obtención de los algoritmos utilizaremos normalmente dos métodos:

Diseño descendente o modular: El problema se divide en subprogramas de más fácil resolución.

Representación gráfica de funciones. La representación como un solo problema es difícil de resolver, sin embargo podemos calcular los cortes con los ejes, máximos, mínimos y puntos de inflexión, asíntotas,... Una vez obtenidos los resultados podemos proceder a juntarlos y obtener la representación gráfica.

Diseño por refinamiento por pasos: Se va buscando la solución y se va especificando hasta llegar a una solución concreta que sí sabemos resolver.

Resolución de la serie de Taylor: $\cos(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + ...$

Solución parcial x^n

Solución parcial n!

Generalmente la utilización de los dos métodos es complementaria.

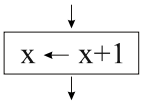
Programación del algoritmo

Una vez determinado el algoritmo hay que escribirlo en un lenguaje de alto nivel. Por eso lo mejor es escribir el algoritmo en un lenguaje restringido que sea fácil de traducir a un lenguaje de alto nivel. Existen dos formas básicas de escribir los algoritmos para conseguir esto: El pseudocódigo y los organigramas (o diagramas de flujo.)

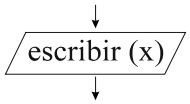
El **pseudocódigo** es una manera de escribir algoritmos de forma poco estricta (con una sintaxis relajada) o estructuras de datos poco detalladas, pero intentando acercar las ideas del algoritmos a estructuras y sintaxis parecidas a las de los lenguajes de alto nivel en los que vamos a programar el 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.

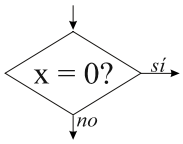
Las instrucciones se representan en rectángulos:



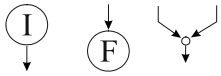
Las entradas y salidas en romboides:



Las condiciones en rombos:



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



Ejercicios:

1. Calcular la tabla del dos.
2. Calcular el cuadrado de los 10 primeros números.
3. Calcular el factorial de un número.

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.)

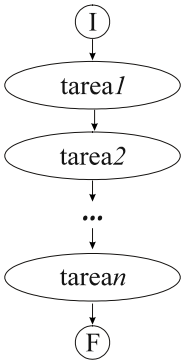
Estructuras secuenciales

En una estructura secuencial una instrucción sigue a otra en una secuencia lineal.

Pseudocódigo

```
Inicio
        tarea1
        tarea2
        ...
        tarean
Fin
```

Organigrama



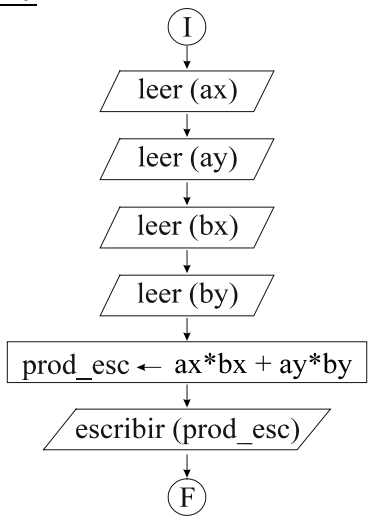
Ejemplo: Producto escalar de dos vectores bidimensionales.

$$(ax, ay) \bullet (bx, by) = ax*bx + ay*by$$

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.

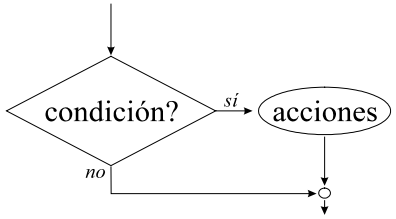
Alternativas simples

Se realiza una acción o conjunto de acciones si se cumple una determinada condición.

Pseudocódigo

```
...
Si ( condición ) entonces
        acciones
Fin_si
...
```

Organigrama

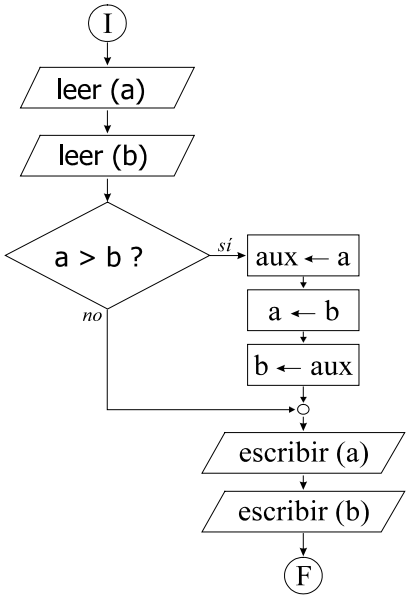


Ejemplo: Ordenar dos números (Leidos dos números escribir por pantalla primero el menor y luego el mayor)

Pseudocódigo

```
Inicio
        Leer (a)
        Leer (b)
        Si (a > b) Entonces
                aux ← a
                a ← b
                b ← aux
        Fin_si
        Escribir (a)
        Escribir (b)
Fin
```

Organigrama



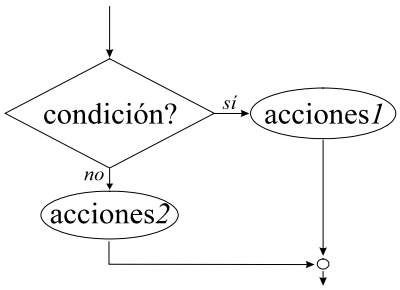
Alternativas dobles

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

Pseudocódigo

```
...
Si ( condición ) entonces
        acciones1
sino
        acciones2
Fin_si
...
```

Organigrama

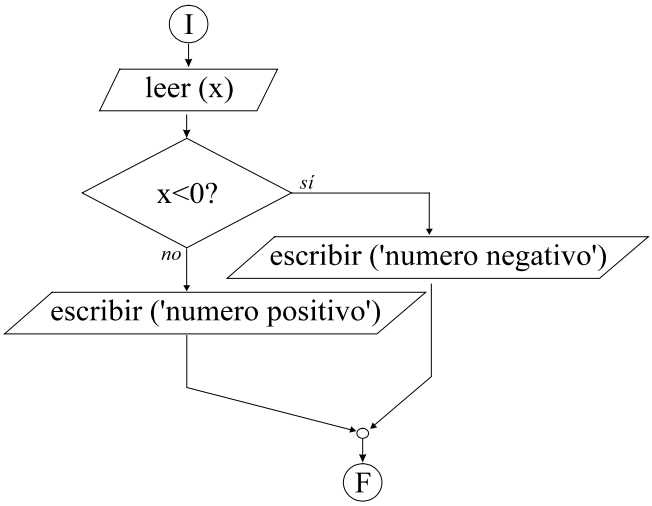


Ejemplo sencillo: Dado un número, decir si es positivo o negativo.

Pseudocódigo

```
Algoritmo Positivo_Negativo
Variables
        x: Entero
Inicio
        Leer (x)
        Si (x<0) entonces
                Escribir ('Numero negativo')
        sino
                Escribir ('Numero positivo')
        Fin_si
Fin
```

Organigrama



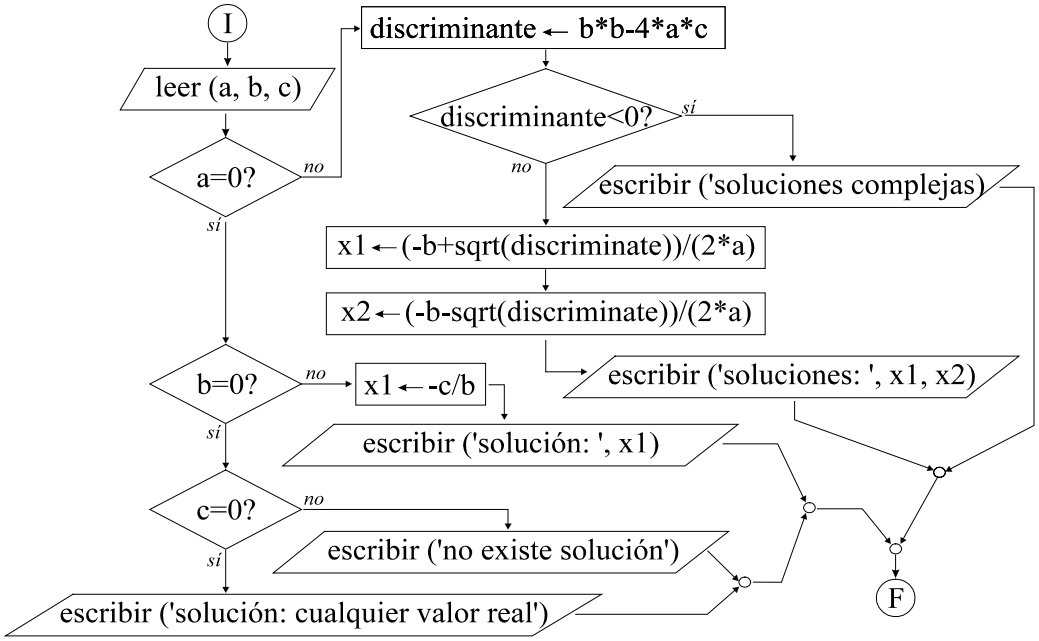
Ejemplo: Cálculo de las soluciones de una ecuación de segundo grado.

$$x = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Pseudocódigo

```
Algoritmo ecuación_segundo_grado
Variables
        a, b, c: entero
        x1, x2: real
        discriminante: entero
Inicio
        Leer (a, b, c)
        Si1 (a = 0) entonces
                Si2 (b = 0) entonces
                        Si3 (c = 0) entonces
                                Escribir ('La solución es cualquier valor real')
                        Sino3
                                Escribir ('No existe ningún valor que cumpla la ecuación')
                        Fin_si3
                Sino2
                        x1 ← -c / b
                        Escribir (x1)
                Fin_si2
        Sino1
                discriminante ← b * b - 4 * a * c
                Si4 (discriminate < 0) entonces
                        Escribir ('Soluciones complejas')
                Sino4
                        x1 ← (-b + sqrt (discriminante) ) / ( 2 * a)
                        x2 ← (-b + sqrt (discriminante) ) / ( 2 * a)
                        Escribir (x1, x2)
                Fin_si4
        Fin_si1
Fin
```

Organigrama



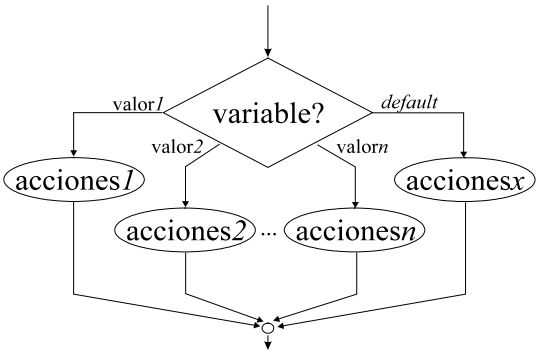
Alternativas múltiples

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: accionesx
Fin_según_sea
...
```

Organigrama



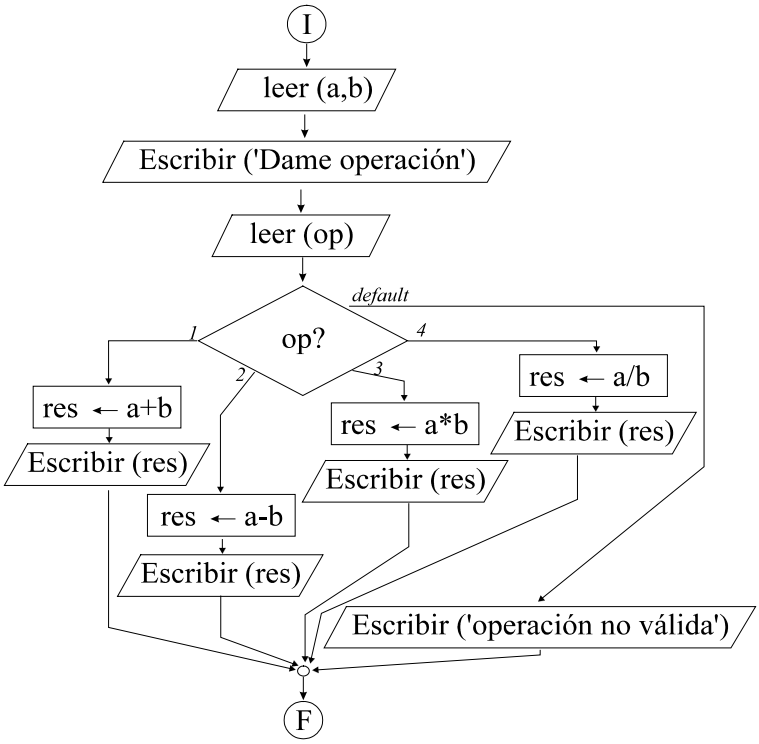
Este tipo de estructuras se utiliza especialmente cuando se programan menús de selección.

Ejemplo: Calculadora infantil. 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

```
Algoritmo Calculadora
Variables
    a, b, op: Entero
    res: Real
Inicio
    Escribir ( 'Dame números')
    Leer (a, b)
    Escribir ('Dame operación:')
    Escribir ('(1-Suma/2-Resta')
    Escribir('(3-Multiplic./4-División')
    Leer (op)
    Según_sea (op) hacer
        Caso 1: res ← a + b
                Escribir (res)
        Caso 2: res ← a - b
                Escribir (res)
        Caso 3: res ← a * b
                Escribir (res)
        Caso 4: res ← a / b
                Escribir (res)
        Default: Escribir ('Operación
                no válida')
    Fin_según_sea
Fin
```

Organigrama



Estructuras repetitivas (o 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 en programación estructurada:

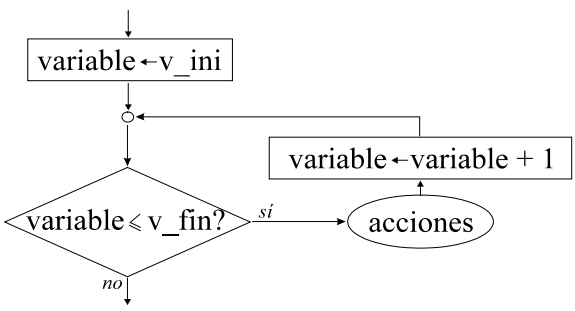
Bucle *Desde…Hasta*

Este bucle se utiliza cuando sabemos el número de veces que queremos que se realice una cierta tarea.

Pseudocódigo

```
...
Desde variable ← v_ini hasta v_fin hacer
        acciones
Fin_desde
...
```

Organigrama



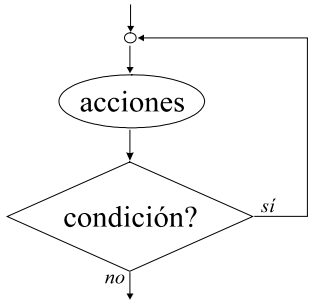
Bucle *Hacer…Mientras*

Este bucle lo utilizaremos 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



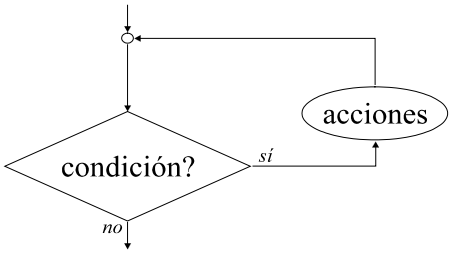
Bucle Mientras…Hacer

Es muy parecido al anterior, pero en este caso la comprobación de la condición se realiza antes de ejecutar la tarea, de manera que la tarea puede no llegar a hacerse.

Pseudocódigo

```
...
Mientras (condición) Hacer
        acciones
Fin_Mientras
...
```

Organigrama



Bucles anidados e independientes.

Existen dos maneras básicas de utilizar varios bucles: De forma anidada y de forma independiente.

De forma independiente nos limitaremos a ir haciendo los bucles de manera que al finalizar uno empezará el siguiente. De esta forma las tareas entre bucles son independientes (cálculo del número combinatorio)

Otra forma es mediante la utilización de bucles anidados. Los 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 (algoritmo que muestre las tablas de multiplicar del 1 al 10.)

Programación modular

Cuando realizamos ciertos programas (por ejemplo el programa de números combinatorios) podemos darnos cuenta que existen ciertas partes del programa que se repiten de forma muy similar, de manera que cambiando pocos elementos quedarían realmente iguales.

Estas partes del programa que se repiten podríamos hacerlas depender de unos valores para que sirviesen para cualquier propósito. A estas partes de los programas es a lo que llamaremos módulos o subprogramas. Con esta separación podemos reutilizar ciertas partes del código de forma más sencilla, y podemos escribir más fácilmente los programas utilizando el diseño descendente para la resolución de algoritmos.

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 y los procedimientos (las funciones devuelven uno o más valores, mientras que los procedimientos devuelven ninguno o más.)

Ambito de las variables

Variables Locales

Son propias de cada módulo y sólo existen mientras dura la ejecución del módulo. Cuando la ejecución del módulo desaparece, las variables locales desaparecen.

Variables Globales

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.

Paso de parámetros

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 si…

…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.

En matemáticas la recursividad es una forma sencilla y habitual de definir una función a partir de ella misma (por ejemplo la definición de factorial o de los números de Fibonacci.)

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.