Tarea: Diagramas de Flujo y seudocódigo

[Escribir el subtítulo del documento]

CFPI: A Carballeira

Módulo: Programación

Contenido

Diagramas de flujo y seudocódigo

Fases de creación de un programa

El proceso de resolución de problemas en un ordenador conduce a la escritura de un programa y su ejecución. Las fases en el desarrollo de un programa pueden resumirse de la siguiente forma:

1. Analizar el problema consiste en conocer perfectamente en qué consiste y qué resultados se desean obtener.
2. Planificación de la resolución del problema, dividiéndolo, si es complicado, en una secuencia de etapas más simples. Esta fase se lleva a cabo EN UN PAPEL, estableciendo lo más claramente posible la finalidad de cada etapa, los datos que se necesitan de entrada, los datos que producirían en salida, los algoritmos que se utilizarán, etc.
3. Edición del código fuente, es decir, escritura del mismo utilizando un editor de textos simple (sin formato) y un lenguaje de programación. Los programas fuente serán almacenados en ficheros de texto, normalmente en el disco duro del ordenador.
4. Compilación y ejecución del programa al lenguaje máquina.
5. Corrección de errores del programa. Los errores se corregirán en el código fuente, repitiendo los pasos 3 y 4 tantas veces como sea necesario. Si se producen errores en la lógica del programa, es decir, si el programa "funciona" pero produce resultados incorrectos, hay que modificar el algoritmo volviendo al paso 2. Estos errores son los más difíciles de detectar.
6. Documentación. Una vez que el programa funcione correctamente, es conveniente revisar el código fuente para ordenarlos, eliminar cálculos innecesarios e incluir las líneas de comentario necesarias, que normalmente deben incluir unas breves explicaciones al principio del código sobre la finalidad del programa y sus argumentos de entrada y de salida.

Algoritmos

Un ordenador es capaz de realizar “sólo” determinadas acciones sencillas, tales como sumar, comparar o transferir datos, pero los problemas que normalmente interesa resolver son más complejos.

Para resolver un problema real es necesario, en primer lugar, encontrar un método de resolución y, posteriormente, determinar la sucesión de acciones sencillas (susceptibles de ser ejecutadas por un ordenador) en que se descompone dicho método.

No todos los métodos de solución de un problema pueden ser puestos es práctica en un ordenador.

Para que un procedimiento pueda ser implantado en un ordenador debe ser:

- Preciso: estar compuesto de pasos bien definidos (no ambiguos) y ordenados.
- Definido: si se sigue dos veces, se obtiene el mismo resultado cada vez.
- Finito: tener un número finito de pasos.

Un procedimiento o método para resolver un problema que cumpla los requisitos anteriores se dice que es un algoritmo. Se puede dar por tanto la siguiente definición:

- Un algoritmo es un método para resolver un problema mediante una secuencia de pasos bien definidos, ordenados y finitos.

Para que se pueda ejecutar el algoritmo es preciso, además, que se disponga de las “herramientas” adecuadas para llevar a cabo cada uno de los pasos. Si no es así, estos deberán, a su vez, ser descompuestos en una secuencia (algoritmo) de pasos más simples que sí se puedan llevar a cabo.

- Un programa de ordenador es una sucesión de ´ordenes que describen un algoritmo, escritas de forma que puedan ser entendidas por el ordenador.

En un algoritmo (y por tanto en un programa) se distinguen las siguientes acciones:

- Entrada: es la información de partida que necesita el algoritmo para arrancar.
- Proceso: es el conjunto de todas las operaciones a realizar.
- Salida: son los resultados obtenidos.

  Ejemplo:

Algoritmo 5.1 Preparar una taza de té.

```
Entrada: tetera, taza, bolsa de té
Salida: taza de té

Inicio
   Tomar la tetera
   Llenarla de agua
   Encender el fuego
   Poner la tetera en el fuego
   Esperar a que hierva el agua
   Tomar la bolsa de té
   Introducirla en la tetera
   Esperar 1 minuto
   Echar el té en la taza
Fin
```

Representación de algoritmos

Las dos herramientas más utilizadas comúnmente para describir algoritmos son:

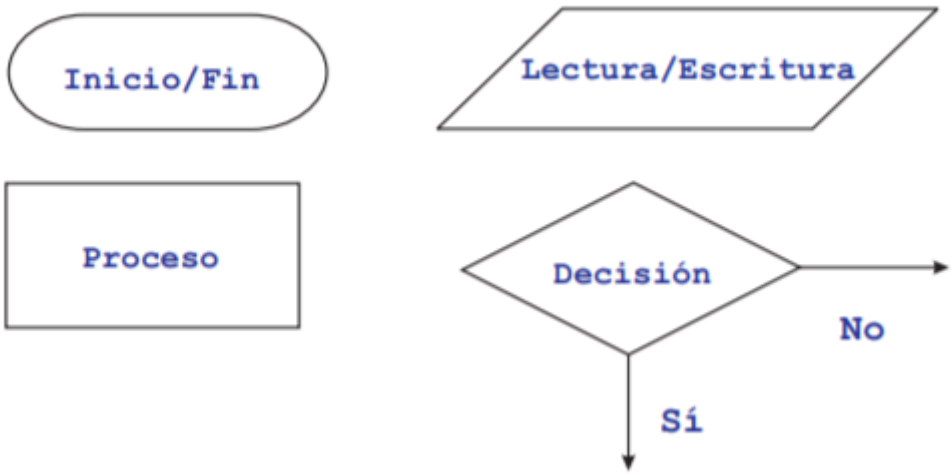
- Diagramas de flujo
- Seudocódigo.

Diagramas de flujo

Diagramas de flujo: son representaciones graficas de secuencias de pasos a realizar. Cada operación se representa mediante un símbolo normalizado el Instituto Norteamericano de Normalización (ANSI - American National Standars Institute). Las líneas de flujo indican el orden de ejecución.

Algunos de los símbolos principales se muestran como son: Inicio/Fin del algoritmo, Lectura/Escritura de datos que el programa necesita o genera (por ejemplo, lectura de datos que se teclean o escritura de datos en un fichero); Proceso conjunto de instrucciones secuenciales; Decisión es una bifurcación en el flujo del algoritmo en base a que se verifique o no cierta condición

Los diagramas de flujo suelen ser usados solo para representar algoritmos pequeños, ya que abarcan mucho espacio.



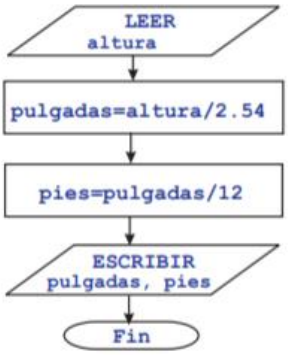
Seudocódigos

Describen un algoritmo de forma similar a un lenguaje de programación pero sin su rigidez, de forma más parecida al lenguaje natural. Presentan la ventaja de ser más compactos que los diagramas de flujo, más fáciles de escribir para las instrucciones complejas y más fáciles de transferir a un lenguaje de programación. El seudocódigo no está regido por ningún estándar.

En estos apuntes usaremos las palabras LEER/IMPRIMIR para representar las acciones de lectura de datos (el programa recibe datos desde algún sitio) y salida de datos (el programa escribe información en algún medio)

Algoritmo 5.2 Calcular una altura en pulgadas (1 pulgada=2.54 cm) y pies (1 pie=12 pulgadas), a partir de la altura en centímetros, que se introduce por el teclado.

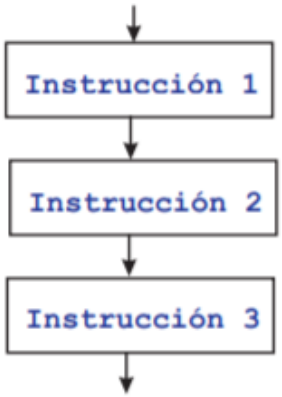
```
Inicio
   1- IMPRIMIR 'Introduce la altura en centimetros: '
   2- LEER: altura
   3- CALCULAR pulgadas=altura/2.54
   4- CALCULAR pies=pulgadas/12
   5- IMPRIMIR 'La altura en pulgadas es: ', pulgadas
   6- IMPRIMIR 'La altura en pies es : ', pies
Fin
```



Estructura secuencial

Es aquella en la que una acción (instrucción) sigue a la otra en el orden en el que están escritas. Su representación y el diagrama de flujo se muestran en la Figura

```
...
Instrucción 1
Instrucción 2
Instrucción 3
...
```



Estructuras de control: condicionales y bucles

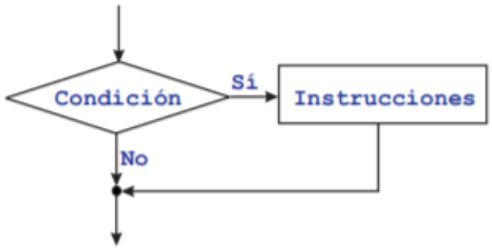
Son parte fundamental de cualquier lenguaje. Sin ellas, las instrucciones de un programa solo podrían ejecutarse en el orden en que están escritas (orden secuencial). Las estructuras de control permiten modificar este orden. Hay dos categorías de estructuras de control: Condicionales o bifurcaciones: permiten que se ejecuten conjuntos distintos de instrucciones, en función de que se verifique o no determinada condición. Bucles o repeticiones: permiten que se ejecute repetidamente un conjunto de instrucciones, bien un número pre-determinado de veces, o bien hasta que se verifique una determinada condición. En términos de un lenguaje de programación, que se verifique o no una condición se traduce en que una (adecuada) expresión lógica tome el valor VERDADERO (TRUE) o tome el valor FALSO (FALSE). En los casos más sencillos y habituales la condición suele ser una comparación entre dos datos, como por ejemplo: si a < b hacer una cosa y en caso contrario hacer otra distinta. A continuación se describen las distintas estructuras de control. Para cada una de ellas se describe el diagrama de flujo y la sintaxis de la sentencia genérica en casi todos los lenguajes de programación. Obsérvese que todas ellas tienen una ´única entrada y una ´única salida

Estructura condicional simple: SI/IF

Este es el tipo más sencillo de estructura condicional. Sirve para implementar acciones condicionales del tipo siguiente:

- Si se verifica una determinada condición, ejecutar una serie de instrucciones y luego seguir adelante.
- Si la condición NO se cumple, NO se ejecutan dichas instrucciones y se sigue adelante.

```
...
if condición
   instrucciones
end
...
```



Obsérvese que, en ambos casos (que se verifique o no la condición), los “caminos” bifurcados se unen posteriormente en un punto, es decir, el flujo del programa recupera su carácter secuencial, y se continua ejecutando por la instrucción siguiente a la estructura SI/IF.

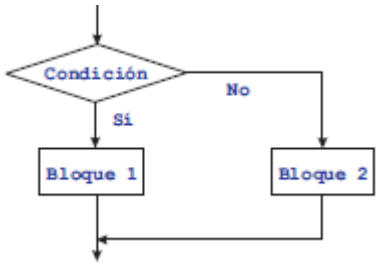
Estructura condicional doble: SI-SI NO/IF- ELSE

Este tipo de estructura permite implementar condicionales en los que hay dos acciones alternativas:

- Si se verifica una determinada condición, ejecutar una serie de instrucciones (bloque 1).
- Si no, esto es, si la condición NO se verifica, ejecutar otra serie de instrucciones (bloque

En otras palabras, en este tipo de estructuras hay una alternativa: se hace una cosa o se hace la otra. En ambos casos, se sigue por la instrucción siguiente a la estructura IF - ELSE.

```
...
if condición
   bloque-1
else
   bloque-2
end
...
```



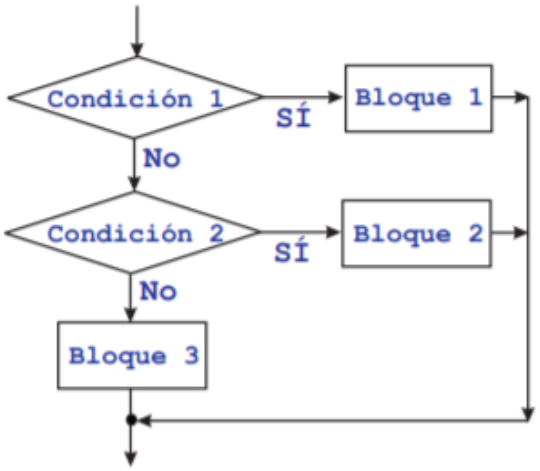
Estructura condicional múltiple: SI-SI NO-SI NO/ IF - ELSEIF - ELSE

En su forma más general, la estructura IF - ELSEIF - ELSE permite implementar condicionales más complicados, en los que se "encadenan" condiciones en la forma siguiente:

- Si se verifica la condición 1, ejecutar las instrucciones del bloque 1.
- Si no se verifica la condición 1, pero S´I se verifica la condición 2, ejecutar las instrucciones del bloque 2.
- Si no, esto es, si no se ha verificado ninguna de las condiciones anteriores, ejecutar las instrucciones del bloque 3.

En cualquiera de los casos, el flujo del programa continua por la instrucción siguiente a la estructura IF - ELSEIF - ELSE.

```
...
if condición-1
   bloque-1
elseif condición-2
   bloque-2
else
   bloque-3
end
...
```



Algoritmo 5.5 Determinación del signo de un número: positivo, negativo o nulo.

```
Inicio
   1- LEER X
   2- Si X>0
        IMPRIMIR 'El número tiene signo positivo'
      Si no, si X<0
        IMPRIMIR 'El numero tiene signo negativo'
      Si no
        IMPRIMIR 'El numero es nulo'
Fin
```

En la estructura IF - ELSEIF - ELSE se puede multiplicar la cláusula ELSEIF, obteniéndose así una "cascada" de condiciones, como se muestra en el organigrama, cuyo funcionamiento es claro.

En este tipo de estructura condicional, la cláusula ELSE junto con su bloque de instrucciones puede no estar presente.

Las distintas estructuras condicionales descritas pueden ser anidadas, es decir, puede incluirse una estructura IF (de cualquier tipo), como parte de las instrucciones que forman el bloque de uno de los casos de otro IF. Como es lógico, no puede haber solapamiento.

Cada estructura IF debe tener su propio fin (en).

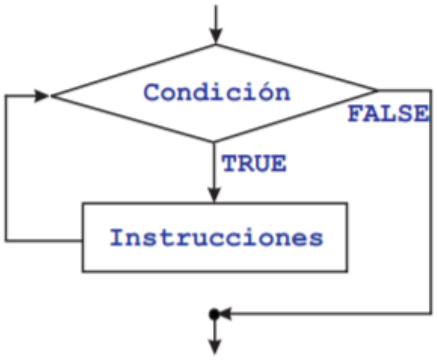
Algoritmo 5.6 Dados dos números reales, **a** y **b**, y el símbolo, **S** (carácter), de un operador aritmético (+, -, *, /), imprimir el resultado de la operación **a S b**

```
Inicio
   LEER a
   LEER b
   LEER S
   Si S='+'
       IMPRIMIR 'El resultado es =', a+b
   Si no, si S='-'
       IMPRIMIR 'El resultado es =', a-b
   Si no, si S='*'
       IMPRIMIR 'El resultado es =', a*b
   Si no, si b=0
       Si a=0
         IMPRIMIR 'El resultado es =', NaN (indeterminación)
       Si no
         IMPRIMIR 'El resultado es =', Inf (infinito)
       Fin Si
   Si no
       IMPRIMIR 'El resultado es =', a/b
   Fin Si
Fin
```

Estructura repetitiva condicional: MIENTRAS/ WHILE

Permite implementar la repetición de un mismo conjunto de instrucciones mientras que se verifique una determinada condición: el número de veces que se repetir 'a el ciclo no está definido a priori.

```
...
while expresión-lógica
   instrucciones
end
...
```



Su funcionamiento es evidente, a la vista del diagrama: 1. Al comienzo de cada iteración se evalúa la expresión lógica. 2. Si el resultado es VERDADERO, se ejecuta el conjunto de instrucciones y se vuelve a iterar, es decir, se repite el paso 1. 3. Si el resultado es FALSO, se

detiene la ejecución del ciclo WHILE y el programa se sigue ejecutando por la instrucción siguiente al END.

Algoritmo 5.11 Imprimir de forma ascendente los 100 primeros números naturales.

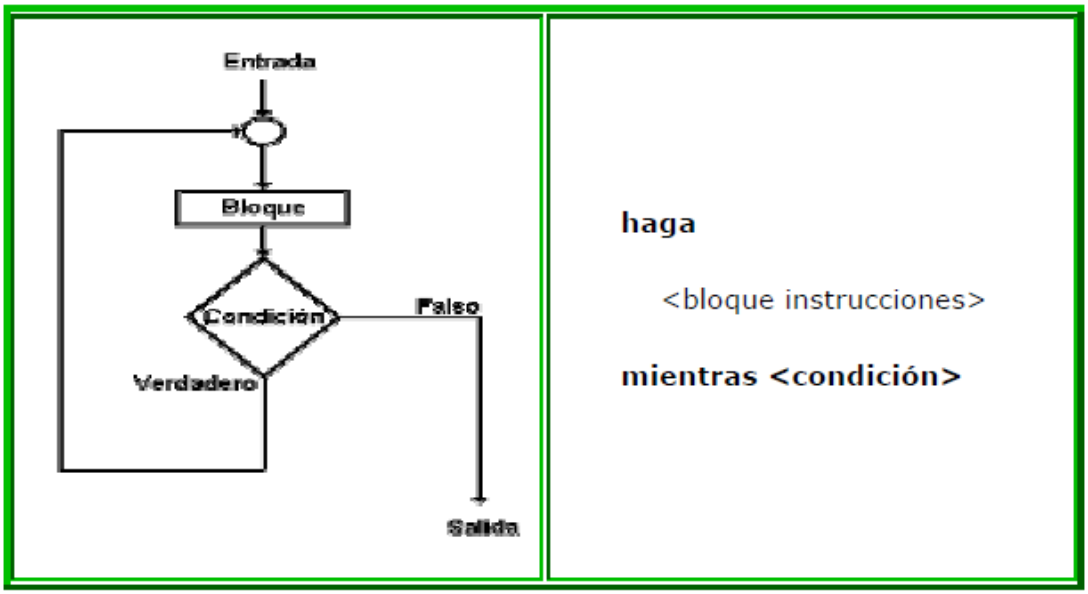
```
Inicio
   i=1
   Mientras que i ≤ 100
      IMPRIMIR i
      HACER i=i+1
   Fin Mientras
Fin
```

Estructura de repetición indexada: HAGA MIENTRAS /DO-WHILE

El ciclo haga-mientras es similar al ciclo mientras, la diferencia radica en el momento de evaluación de la condición.

En el ciclo haga-mientras la condición se evalúa después de ejecutar el bloque de instrucciones, por lo tanto, el bloque se ejecuta por lo menos una vez. Este bloque se ejecuta nuevamente si la condición evalúa a verdadero, y no se ejecuta más si se evalúa como falso.

La forma general del ciclo haga-mientras es la siguiente:



Donde, <bloque instrucciones> es el conjunto de instrucciones que se ejecuta y <condición> es la expresión lógica que determina si el bloque se ejecuta. Si la <condición> se evalúa como verdadero el bloque es ejecutado de nuevo y si es evaluada como falso no es ejecutado.

Después de ejecutar el bloque de acciones se evalúa la <condición>.

Ejemplos

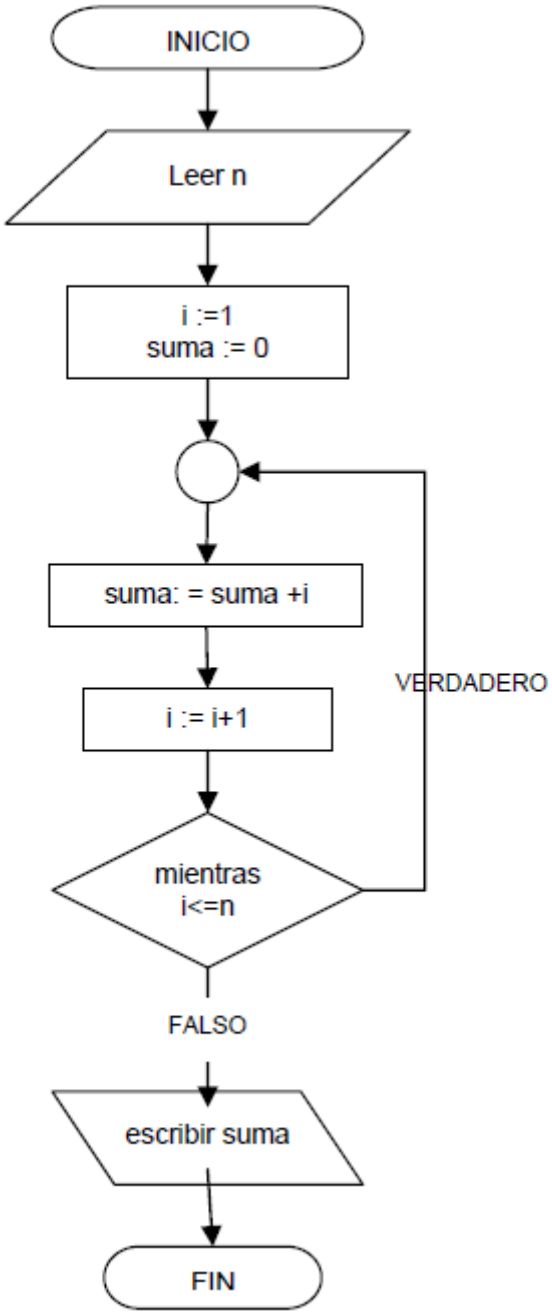
Ejemplo 1. El problema de calcular la suma de los números naturales desde 1 hasta n, se puede solucionar usando el ciclo haga-mientras. A continuación se describe el algoritmo solución:

```
escribir ( "Ingrese el número: " )
leer (n)  /* lee el primer número */
suma :=0 /* inicia la suma en cero */

i :=1 /* empieza la variable que recorre los números en 1 */

haga
   suma := suma + i /* en cada iteración suma el número i */
   i := i + 1 /* incrementa i en 1 para tomar el siguiente número en la próxima iteración */
mientras (i <= n)

escribir ( "La suma es: ", suma )
```


INICIO
Leer n
i :=1
suma := 0
suma: = suma +i
i := i+1
mientras
i<=n
VERDADERO
FALSO
escribir suma
FIN

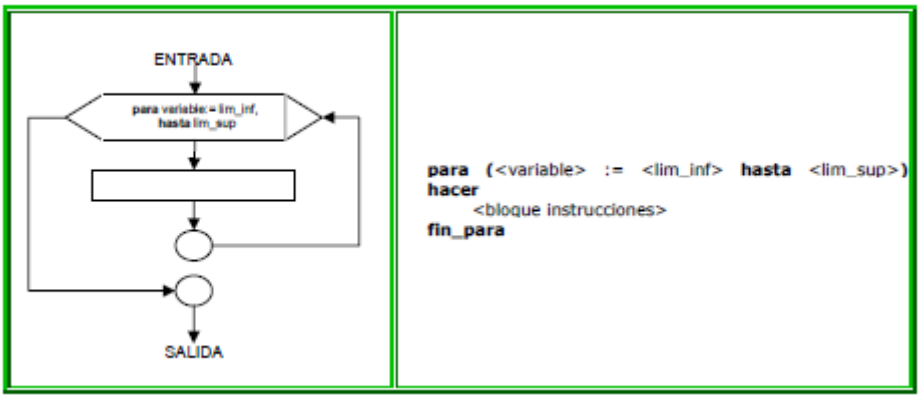
LINEA	n	i	suma	ENTRADA	SALIDA
4					Ingrese el número:
5	5			5	
6			0		
7		1			
8	Se ejecuta la instrucción **haga**, es decir pasa a la línea 9				
9			1		
10		2			
11	La condición es evaluada a verdadero, por lo tanto se salta hasta la línea que contiene la condición del ciclo **haga_mientras** en ejecución, es decir, hasta la línea 8				
8	Se ejecuta la instrucción **haga**, es decir pasa a la línea 9				
9			3		
10		3			
11	La condición es evaluada a verdadero, por lo tanto se salta hasta la línea que contiene la condición del ciclo **haga_mientras** en ejecución, es decir, hasta la línea 8				
8	Se ejecuta la instrucción **haga**, es decir pasa a la línea 9				
9			6		
10		4			
11	La condición es evaluada a verdadero, por lo tanto se salta hasta la línea que contiene la condición del ciclo **haga_mientras** en ejecución, es decir, hasta la línea 8				
8	Se ejecuta la instrucción **haga**, es decir pasa a la línea 9				
9			10		
10		5			
11	La condición es evaluada a verdadero, por lo tanto se salta hasta la línea que contiene la condición del ciclo **haga_mientras** en ejecución, es decir, hasta la línea 8				
8	Se ejecuta la instrucción **haga**, es decir pasa a la línea 9				
9			15		
10		6			
11	La condición es evaluada a falso, por lo tanto este ciclo termina y se salta a la línea 12				
12					La suma es: 15

Estructura de repetición indexada: PARA /FOR

El **ciclo para** ejecuta un bloque de instrucciones un número determinado de veces. Este número de veces está determinado por una *variable contadora* (de tipo entero) que toma valores desde un *límite inferior* hasta un *límite superior*. En cada ciclo después de ejecutar el bloque de instrucciones, la *variable contadora* es incrementada en 1 automáticamente y en el momento en que la variable sobrepasa el *límite superior* el ciclo termina.

El valor final de la *variable contadora* depende del lenguaje de programación utilizado, por lo tanto, no es recomendable diseñar algoritmos que utilicen el valor de la *variable contadora* de un ciclo **para**, después de ejecutar el mismo. De la definición de **ciclo para** se puede inferir que el bloque de instrucciones no se ejecuta si el *límite inferior* es mayor al *límite superior*.

La forma general del ciclo para es la siguiente:.



Donde <variable> es la variable contadora del ciclo, la cual debe ser de tipo entero. <lim_inf> es el valor inicial que toma la variable contadora. <lim_sup> es el último valor que toma la variable contadora; cuando el valor de la variable contadora supere este valor, el ciclo termina.

<bloque instrucciones> es el conjunto de instrucciones que se ejecuta en cada iteración, mientras la variable contadora no sobrepase el <lim_sup>.

Casos:

Cuando <lim_inf> es menor que <lim_sup> ocurre lo siguiente:

1. La variable contadora se vuelve igual a <lim_inf>
2. Se ejecuta <bloque de instrucciones>
3. Se incrementa automáticamente en 1 la variable contadora del ciclo.
4. Si el valor de contador del ciclo es menor o igual que <lim_sup> se vuelve de nuevo al paso 2. De otro modo se abandona el ciclo.

Es de anotar que el valor final de la variable contadora queda incrementada por encima del <lim_sup> para que pueda finalizar el ciclo

Cuando <lim_inf> es mayor que <lim_sup> el ciclo termina sin ejecutarse nunca el <bloque de instrucciones>. Tenga en cuenta que no se genera error al correr el programa

Ejemplo:

para (x:=5 hasta 4) hacer.

Esta línea de código nunca se ejecuta.

Es de anotar que los lenguajes de programación tienen una variable a esta instrucción para que el valor pueda ir en descenso.

Tanto <lim_inf> como <lim_sup> pueden ser expresiones como en el siguiente

ejemplo:

para(j:=x+1 hasta 2*y) hacer.

En este caso se calculan primero los valores de las expresiones (x+1) y (2*y) empleando para esto los valores actuales de x y y para utilizarlos como <lim_inf> y <lim_sup> respectivamente.

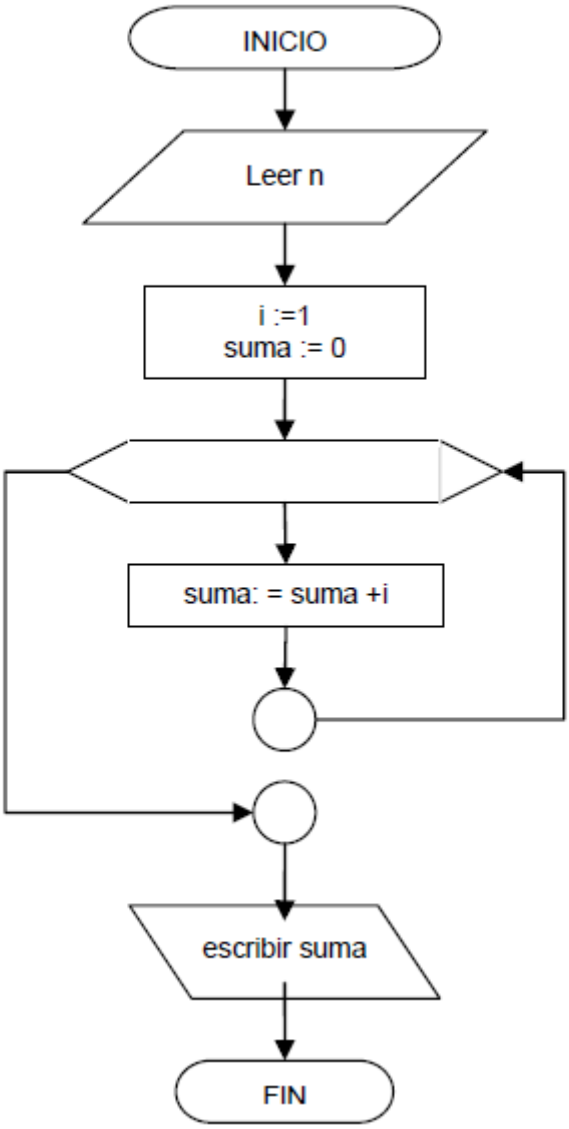
Ejemplo El problema de calcular la suma de los números naturales desde 1 hasta n , se puede solucionar usando el ciclo para, a continuación se muestra el algoritmo solución:

```
escribir("ingrese el número:")
leer n /* lee el primer número */

suma:= 0

para(i :=1 hasta n)hacer
    suma :=suma + i
fin_para

escribir ("La suma es:", suma)
```



Nótese que se requieren menos instrucciones que en las anteriores estructuras dado que el incremento de **i** se hace automáticamente

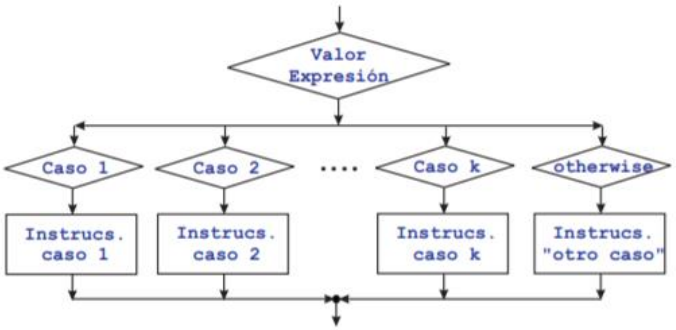
LINEA	N	I	SUMA	ENTRADA	SALIDA
4					Ingrese el número a calcularle la tabla de multiplicar:
5	3			3	
6			0		
Como es la primera vez que llega a esta línea, se asigna en la variable contadora el límite inferior. Ahora se comprueba si la variable contadora es menor o igual al límite superior. Como se ve en la línea 7, en este caso es cierto entonces se ejecuta el bloque de acciones del ciclo **para**, es decir, se pasa a la línea 8.					
7		1			
8			1		
9	Se vuelve a la línea de inicio del ciclo **para**, es decir, línea 7.				
7		2			
Se comprueba si la variable contadora es menor o igual al límite superior. En este caso es cierto entonces se ejecuta el bloque de acciones del ciclo **para**, es decir, se pasa a la línea 8.					
8			3		
9	Se vuelve a la línea de inicio del ciclo **para**, es decir, línea 7.				
7		3			
Se incrementa la variable contadora y se vuelve a la línea de inicio del ciclo para, es decir, línea 7. Como la variable contadora es menor que el límite superior se pasa a la línea 8					

LINEA	N	I	SUMA	ENTRADA	SALIDA
8			6		
9	Se vuelve a la línea de inicio del ciclo **para**, es decir, línea 7.				
7		4			
Se incrementa la variable contadora y se vuelve a la línea de inicio del ciclo para, es decir, línea 6.					
6	La variable contadora no es menor que el límite superior se pasa a la línea siguiente al **fin_para**, es decir, a la línea 10.				
10					La suma es : 3

Estructura de elección entre varios casos: CASO /SWITCH

Este tipo de estructura permite decidir entre varios caminos posibles, en función del valor que tome una determinada instrucción.

```
switch expresión
  case valor-1
    instrucciones caso 1
  case valor-2
    instrucciones caso 2
  ...
  case {valores...}
    instrucciones caso k
  otherwise
    instrucciones
end
```



En cada uno de los casos, el valor correspondiente puede ser o bien un solo valor, o bien un conjunto de valores, en cuyo caso se indican entre llaves. La cláusula OTHERWISE y su correspondiente conjunto de instrucciones pueden no estar presente. El funcionamiento es el siguiente:

1. Al comienzo se evalúa la expresión.

2. Si expresión toma el valor (´o valores) especificados junto a la primera clausula CASE, se ejecuta el conjunto de instrucciones de este caso y después se abandona la estructura SWITCH, continuando por la instrucción siguiente al END.
3. Se repite el procedimiento anterior, de forma ordenada, para cada una de las clausulas CASE que siguen.
4. Si la cláusula OTHERWISE está presente y la expresión no ha tomado ninguno de los valores anteriormente especificados, se ejecuta el conjunto de instrucciones correspondiente.

Obsérvese que se ejecuta, como máximo el conjunto de instrucciones de uno de los casos, es decir, una vez que se ha verificado un caso y se ha ejecutado su conjunto de instrucciones, no se testea el resto de casos, ya que se abandona la estructura. Obviamente, si la cláusula OTHERWISE no está presente, puede ocurrir que no se dé ninguno de los casos. Como ejemplo de utilización, se presenta el mismo proceso del Algoritmo 5.6, pero utilizando la sentencia SWITCH.

Algoritmo 5.12 Dados dos números reales, **a** y **b**, y el símbolo, **S** (carácter), de un operador aritmético (+, -, *, /), imprimir el resultado de la operación **a S b**

```
LEER a , b , S
Elegir caso S
  Caso '+'
     IMPRIMIR 'El resultado es =', a+b
  Caso '-'
     IMPRIMIR 'El resultado es =', a-b
  Caso '*'
     IMPRIMIR 'El resultado es =', a*b
  Caso '*'
     IMPRIMIR 'El resultado es =', a*b
  Caso '/'
     Si a≠0
        IMPRIMIR 'El resultado es =', a/b
     Si no, si b≠0
        IMPRIMIR 'El resultado es =', Inf (infinito)
     Si no
        IMPRIMIR 'El resultado es =', NaN (indeterminación)
     Fin Si
  En otro caso
     IMPRIMIR 'El operador no se reconoce'
  Fin Elegir caso
```