

# Capítulo 04

## Estructuras de Control

Samuel Sepúlveda Cuevas

Mauricio Diéguez Rebolledo

Ania Cravero Leal

Departamento de Ingeniería de Sistemas  
Facultad de Ingeniería, Ciencias y Administración

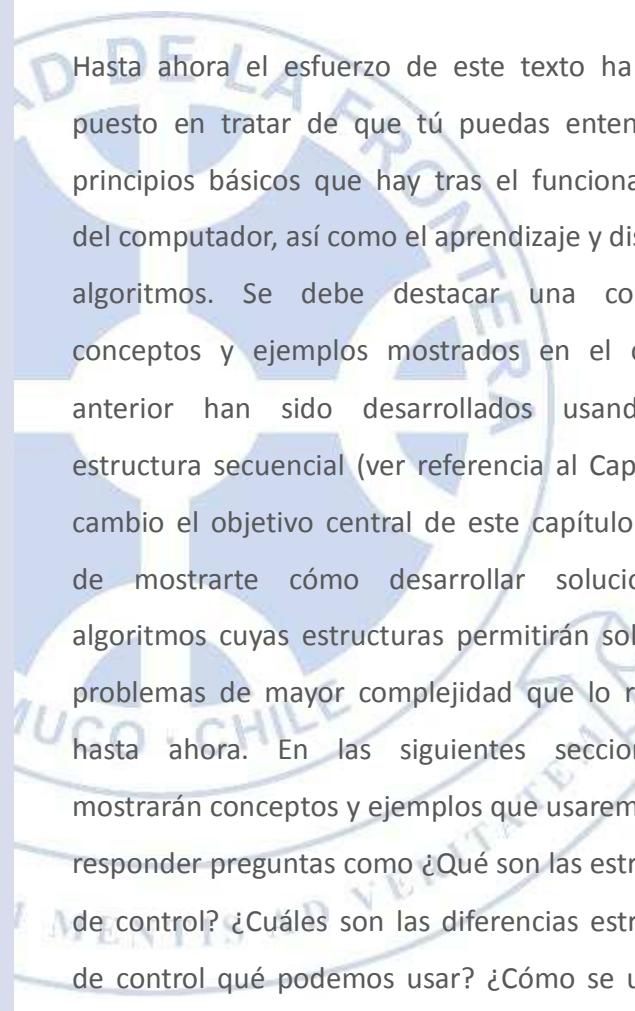
“Proyecto financiado por el Fondo de Desarrollo Educativo de  
la Facultad de Ingeniería, Ciencias y Administración de la  
Universidad de La Frontera”

Versión  
0.9

**TEMARIO**

- 4.1** Introducción
- 4.2** Estructura secuencial
- 4.3** Estructuras de selección
- 4.4** Estructuras de iteración
- 4.5** Ejercicios resueltos
- 4.6** Ejercicios propuestos
- 4.7** Comentarios finales
- 4.8** Referencias

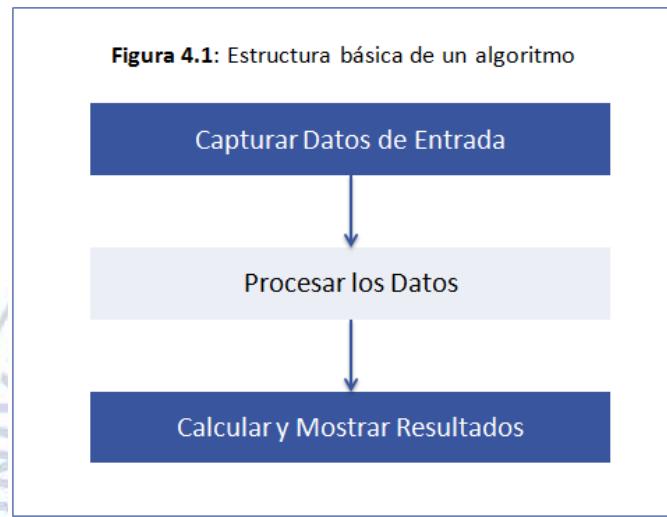
# Estructuras de Control



Hasta ahora el esfuerzo de este texto ha estado puesto en tratar de que tú puedas entender los principios básicos que hay tras el funcionamiento del computador, así como el aprendizaje y diseño de algoritmos. Se debe destacar una cosa, los conceptos y ejemplos mostrados en el capítulo anterior han sido desarrollados usando una estructura secuencial (ver referencia al Cap. 3), en cambio el objetivo central de este capítulo será el de mostrarte cómo desarrollar soluciones y algoritmos cuyas estructuras permitirán solucionar problemas de mayor complejidad que lo realizado hasta ahora. En las siguientes secciones se mostrarán conceptos y ejemplos que usaremos para responder preguntas como ¿Qué son las estructuras de control? ¿Cuáles son las diferencias estructuras de control qué podemos usar? ¿Cómo se usan las estructuras de control?

## 4.1 Introducción

Los algoritmos de los problemas revisados hasta ahora obedecen básicamente a la siguiente estructura:



Esta estructura ha demostrado ser eficiente en los problemas de tipo aritmético-matemático simples revisados hasta ahora. P. ej: calcular áreas de figuras geométricas, obtener promedios, etc.

Pero ¿qué sucede si se desea resolver algo como lo siguiente?

### Ejemplo 4.1:

Se ingresan 2 números por teclado, pero si el segundo número es cero, entonces muestre el primer número en pantalla, a continuación divida el primero por el segundo.

#### Análisis

El usuario del programa debe ingresar dos números y determinar si el segundo número es cero. Luego se debe realizar algunas acciones con cada número ingresado.

Así tenemos que las entradas son los dos números. Pero en cuanto a la salida, tenemos dos posibles, la primera cuando el segundo número ingresado es cero, y la segunda, cuando el segundo número ingresado no es cero.

**Entradas:** dos números (variables numero1, numero2)  
**Salidas:** Si el segundo número es cero, la salida será mostrar el primer número. Si no, no hay salida.

#### Diseño

Proceso: Los pasos a seguir no son lineales, como vimos en el capítulo 3. ¿Cómo solucionamos este problema?

#### Solución

Hacer uso de las **estructuras de control**, que permiten dirigir el flujo de acción dentro del programa, en base a la evaluación del cumplimiento de ciertas condiciones lógicas.

**Ejemplo 4.2:**

Algoritmo en seudocódigo que suma dos números

```
Inicio  
    Ingresar num1, num2  
    Suma ← num1 + num2  
    Escribir suma  
Fin
```

Para poder sumar los 2 números y mostrar el resultado, se debe pasar por todas las instrucciones sin excepción.

## 4.2 Estructura secuencial

Tal como viste en el Cap. 3, una estructura de control secuencial implica que una secuencia de instrucciones se ejecutan secuencialmente, una tras otra sin excepción, desde el principio hasta el final.

Para ilustrar esta idea y así puedas luego comparar con lo mostrado en la sección 4.3, te presentamos el siguiente ejemplo, en el cual se desean sumar 2 números enteros y mostrar el resultado. Este problema puede ser solucionado sin mayores dificultades usando una *Estructura Secuencial*, pues las instrucciones se van ejecutando sucesivamente, siguiendo el orden de aparición de éstas. No hay saltos.

## 4.3 Estructuras de selección

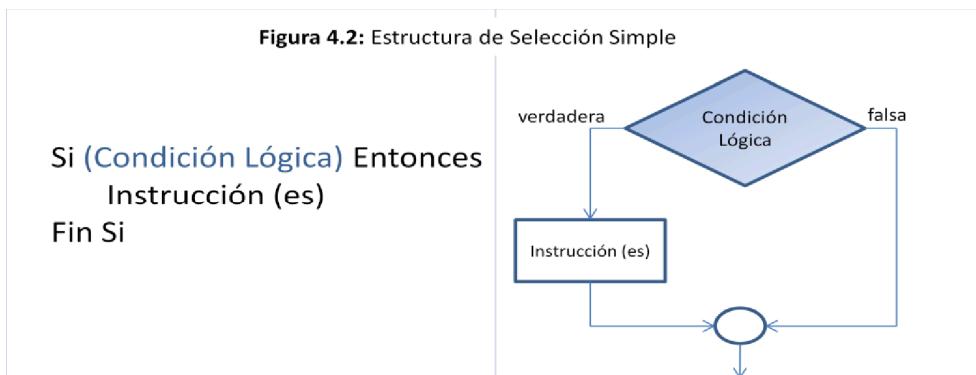
Una ejecución condicional consiste en la ejecución de una (o más) instrucción(es) dependiendo de la evaluación de una condición.

Una estructura condicional es una estructura que permite una ejecución condicional. Existen tres tipos: Simple, Doble y Múltiple.

### 4.3.1 Estructura de selección simple

El caso más simple de una estructura de control de flujo, donde se ejecutan una o más instrucciones, siempre y cuando se cumpla la condición lógica.

Genéricamente a esta expresión se le conoce como estructura o bloque SI, cuya forma típica es:



Entendiendo como funciona la estructura SI

1. Se evalúa la condición y si ésta se cumple, entra al bloque
  - a. se ejecutan las instrucciones que se encuentra en su interior
  - b. luego de ejecutadas las instrucciones, el programa sigue con su secuencia, ejecutando la instrucción que se encuentre a continuación del fin del SI.
2. Si la condición no se cumple, no entra al bloque y el programa sigue con su secuencia, ejecutando la instrucción que se encuentre a continuación del fin del SI.

#### Ejemplo 4.2:

Si recordamos el ej. 4.1, vemos que usar la estructura SI resulta adecuada para resolverlo, con lo cual la solución sería:

##### Análisis:

Se debe ingresar dos números, si el segundo número es cero, entonces se realiza acciones con uno de los números, en caso contrario no se hace nada.

<b>Entradas:</b>	dos números (variables numero1, numero2)
<b>Salidas:</b>	Si el segundo número es cero, la salida será mostrar el primer número Si no, no hay salida.

##### Diseño:

Proceso: Debemos utilizar una estructura de control de selección simple para solucionar el problema, dado que existen dos posibles alternativas de ejecución.

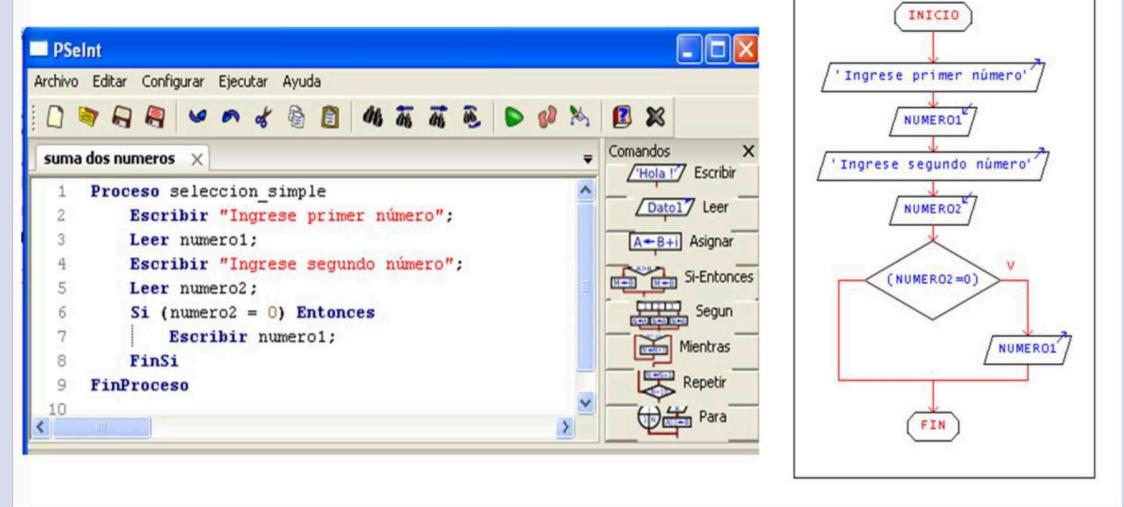
La condición lógica a evaluar para este caso sería:

```

Si (numero2 es igual a cero) Entonces
  Mostrar el primer número por pantalla
Fin Si
  
```

La Figura siguiente muestra la solución en seudocódigo y diagrama de flujo obtenida desde la herramienta PSelnt.

**Figura 4.3:** Solución en seudocódigo y diagrama de flujo



#### 4.3.2 Estructura de selección doble

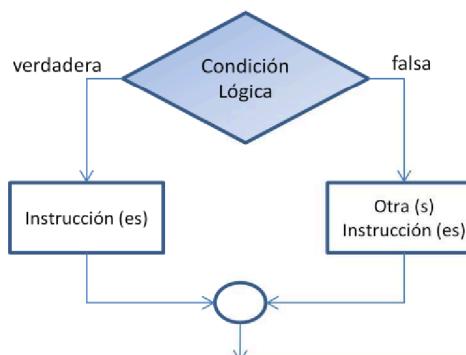
A partir del ejemplo anterior, se puede usar una extensión del ciclo SI, que consiste en evaluar una condición y ejecutar una acción u otra de las propuestas dentro del bloque.

Genéricamente a esta expresión se le conoce como estructura o bloque SI...SINO, cuya forma típica es:

**Figura 4.4:** Estructura de Selección Doble

```

Si (se cumple Condición) Entonces
  Instrucción (es)
Sino
  Otra (s) Instrucción (es)
Fin Si
  
```



### Entendiendo como funciona la estructura SI...SINO

1. Se evalúa la condición y si ésta se cumple, entra al bloque SI
  - a. se ejecutan las instrucciones que se encuentra en el interior
  - b. luego de ejecutadas las instrucciones, el programa sigue con su secuencia, ejecutando la instrucción que se encuentre a continuación del fin del SI...SINO
2. En caso contrario
  - a. se ejecutan la(s) instrucción(es) al interior del SINO
  - b. luego de ejecutadas las instrucciones, el programa sigue con su secuencia, ejecutando la instrucción que se encuentre a continuación del fin del SI...SINO

#### Ejemplo 4.3:

Si se desea construir un algoritmo que permita encontrar y mostrar por pantalla el mayor de 2 números que ingresa el usuario. ¿Cómo lo solucionamos usando la estructura SI...SINO?

##### Análisis:

Se debe mostrar el mayor de dos números. Para ello, el usuario debe ingresar primero los dos números, luego se debe comparar ambos números para obtener el mayor de ellos.

**Entradas:** dos números (variables numero1, numero2)  
**Salidas:** Mostrar el mayor de ambos números.

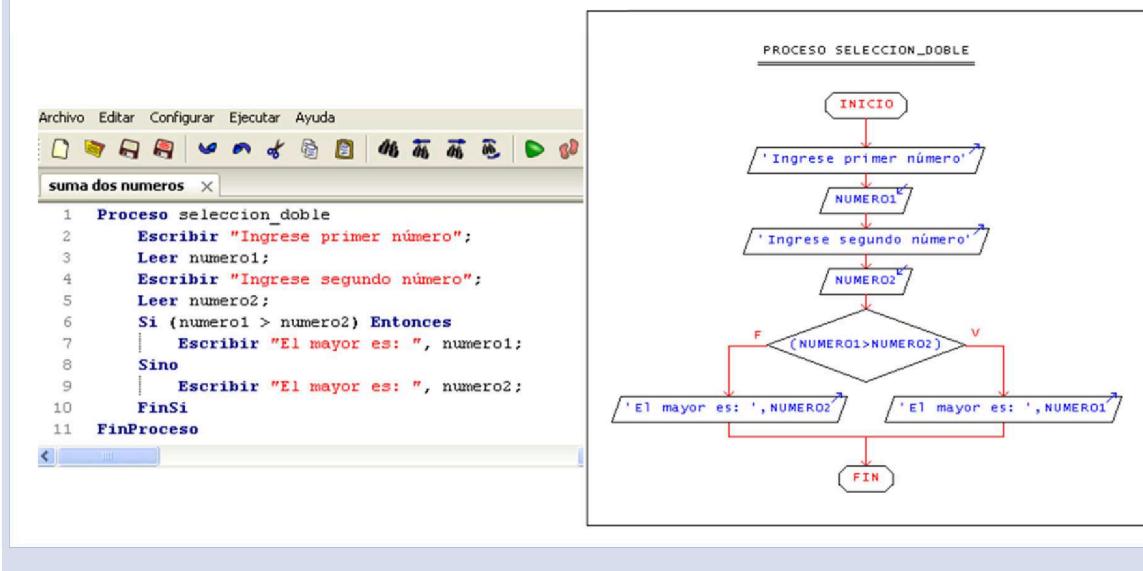
##### Diseño:

Proceso: Para seleccionar el mayor de los dos números debemos utilizar una estructura de selección doble, ya que disponemos de dos posibilidades: que el primer número ingresado sea el mayor, o que el segundo número sea el mayor. Para comparar ambos números debemos utilizar una condición lógica como la que sigue:

```
Si (numero1 > numero2) Entonces
    Mostrar el numero1 por pantalla
Sino
    Mostrar el numero2 por pantalla
Fin Si
```

La figura siguiente muestra la solución en seudocódigo y diagrama de flujo.

Figura 4.5: Solución en seudocódigo y diagrama de flujo



#### 4.3.3 Estructura de selección anidadas

Si bien no corresponde a una categoría en sí misma, las selecciones anidadadas suelen darse con bastante frecuencia cuando comenzamos a solucionar problemas cuya complejidad nos hace evaluar varias condiciones.

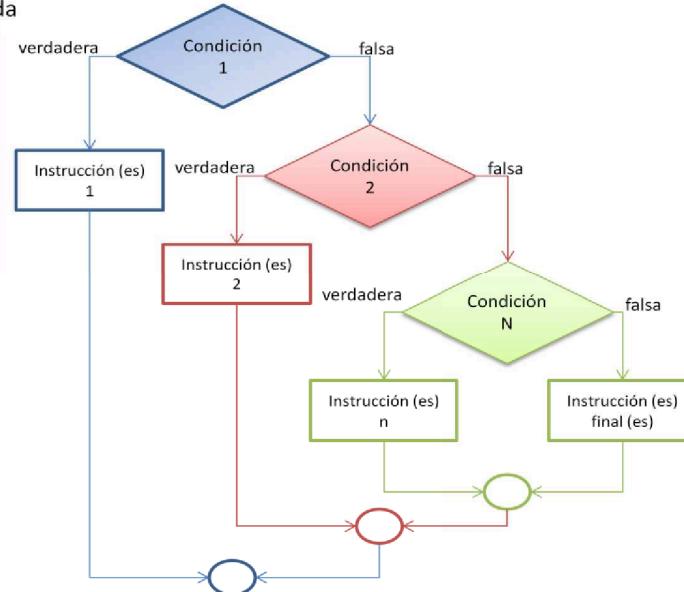
Al igual que la sentencia SI...SINO es una extensión del caso más simple de la sentencia SI...Consiste en un conjunto de opciones que termina con una opción en caso de no haberse cumplido ninguna de las anteriores. Se le conoce como los SI...SINO anidados.

A continuación se muestra un caso en el cual se tienen varias estructuras del tipo SI y SI...SINO, unas contenidas dentro de otras, lo que produce aquello que llamamos anidamiento.

**Figura 4.6:** Estructura selectiva anidada

```

Si (condición1) Entonces
    Instrucción (es) 1
Sino
    Si (condición2) Entonces
        Instrucción (es) 2
    Sino
        Si (condición N) Entonces
            Instrucción (es) n
        Sino
            Instrucción (es) final (es)
        Fin Si
    Fin Si
Fin Si
  
```

**Ejemplo 4.4:**

Construya un algoritmo que dé una solución al siguiente problema. Dada una NOTA que ingresa el profesor entre 1.0 y 7.0, el algoritmo debe transformarla a una nota conceptual según la lista mostrada a continuación:

Si la nota está entre:

- 1,0 y 3,9 → Insuficiente
- 4,0 y 4,9 → Suficiente
- 5,0 y 5,9 → Bien
- 6,0 y 7,0 → Muy bien

**Análisis:**

Se pide que el usuario ingrese una calificación, que corresponde a un número de 1,0 a 7,0. La salida será una nota conceptual de acuerdo a los rangos indicados en el problema.

**Entradas:**

Una nota (variable nota)

**Salidas:**

Mostrar la nota conceptual que corresponde, que puede ser una de las siguientes posibilidades: Insuficiente, Suficiente, Bien, Muy bien

**Diseño:**

Proceso: Se necesitará una estructura de control SI-SINO ya que se tiene varias posibilidades para la salida en pantalla. Ello dependerá de la nota ingresada y al rango que le corresponde. Las condiciones lógicas deben ser las siguientes:

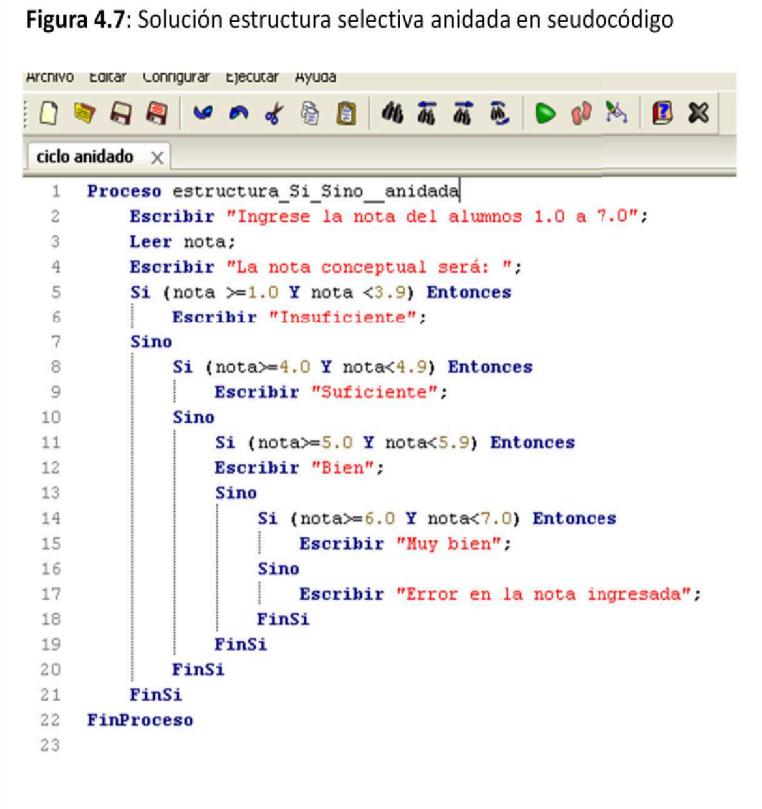
```

Si (nota está entre 1,0 y 3,9) entonces
    Mostrar nota conceptual Insuficiente
Sino
    Si (nota está entre 4,0 y 4,9) entonces
        Mostrar nota conceptual Suficiente
    Sino
        Si (nota está entre 5,0 y 5,9) entonces
            Mostrar nota conceptual Bien
        Sino
  
```

```
Si (nota está entre 6,0 y 7,0) entonces
    Mostrar nota conceptual Muy bien
Fin Si
Fin Si
Fin Si
```

Las figuras siguientes muestran la solución completa del algoritmo en seudocódigo y diagrama de flujo.

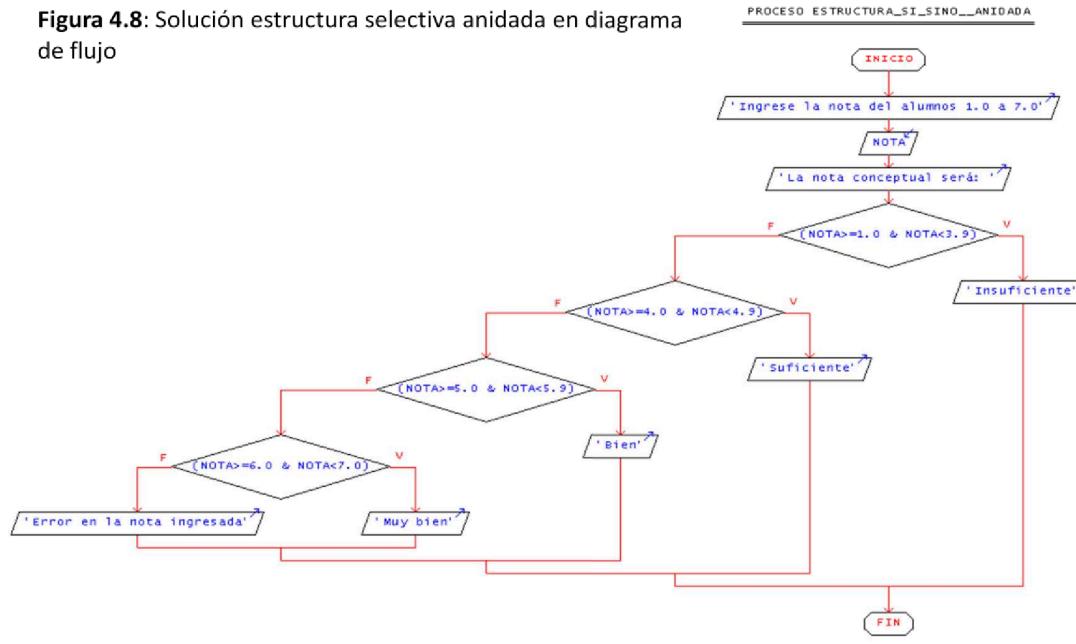
Figura 4.7: Solución estructura selectiva anidada en seudocódigo



The screenshot shows a software window titled "ciclo anidado X". The menu bar includes "Archivo", "Editar", "Configurar", "Ejecutar", and "Ayuda". Below the menu is a toolbar with various icons. The main area contains the following pseudocode:

```
1 Proceso estructura_Si_Sino_anidada
2   Escribir "Ingrese la nota del alumnos 1.0 a 7.0";
3   Leer nota;
4   Escribir "La nota conceptual será: ";
5   Si (nota >=1.0 Y nota <3.9) Entonces
6     Escribir "Insuficiente";
7   Sino
8     Si (nota>=4.0 Y nota<4.9) Entonces
9       Escribir "Suficiente";
10    Sino
11      Si (nota>=5.0 Y nota<5.9) Entonces
12        Escribir "Bien";
13        Sino
14          Si (nota>=6.0 Y nota<7.0) Entonces
15            Escribir "Muy bien";
16            Sino
17              Escribir "Error en la nota ingresada";
18            FinSi
19        FinSi
20    FinSi
21  FinSi
22 FinProceso
23
```

**Figura 4.8:** Solución estructura selectiva anidada en diagrama de flujo



Entendiendo como funciona la estructura SI...SINO anidado

1. Se evalúa si la NOTA está en el rango de 1,0 y 3,9, de ser así se imprime la palabra *Insuficiente*
2. Si al evaluar NOTA, ésta no está en el rango de 1,0 y 3,9, se comienza a ejecutar el bloque SINO
3. Luego se evalúa NOTA para saber si está en el rango de 4,0 y 4,9, de ser verdadero se imprime la palabra *Suficiente*
4. Si no se cumple lo anterior se ingresa en el bloque SINO
5. Luego se evalúa NOTA para saber si está en el rango de 5,0 y 5,9, de ser verdadero se imprime la palabra *Bien*
6. De no cumplirse lo anterior, se evalúa NOTA para saber si está en el rango de 6,0 y 7,0, que de ser verdadero se imprime la palabra *Muy Bien*
7. En caso de no ser verdadero, se imprime *Error*, pues la nota ingresada no está entre 1,0 y 7,0.

Otra cosa importante a poner atención en el ejemplo anterior, es que debes verificar como se van anidando los bloques SI...SINO, con lo cual puedes ver que es muy fácil que una solución se vuelva bastante difícil de entender, por lo cual es muy importante recordar las buenas prácticas de programación vistas en el Cap. 2

(hacer link). Además deberías fijarte en cómo se van cerrando c/u de los bloques SI...SINO con los correspondientes Fin Si.

Dentro de un bloque SI...SINO, puede incluirse otra estructura condicional, anidándose tanto como permita el lenguaje de programación usado. En programas más complejos, es común que se tengan estructuras de selección anidadas, lo importante es saber cuándo se ejecuta un conjunto específico de instrucciones dentro de un SI...SINO anidado.

#### Ejemplo 4.5:

A continuación te presentamos un ejemplo el cual está compuesto por un set de estructuras SI...SINO anidados. Revísalo con atención, fíjate con cuidado dónde comienza y termina cada estructura de selección. Encontrarás a continuación una pregunta para verificar si lo entendiste de forma correcta.

```
Si (condición_01) Entonces
    instrucciones_01;
    Si (condición_02) Entonces
        instrucciones_02;
    Sino
        instrucciones_03;
    Fin Si
    instrucciones_04;
Sino
    instrucciones_05;
Fin Si
```

#### ¿Cuándo se ejecuta cada conjunto de instrucciones?

La respuesta a esta pregunta está resumida en la siguiente tabla, dónde dependiendo del valor lógico de cada condición, se podrán ejecutar los diferentes bloques de instrucciones.

**Tabla 4.1:** Instrucciones que se ejecutan dependiendo de las condiciones lógicas

Se ejecutan las siguientes instrucciones Si	condición_01	condición_02
Instrucciones_01	true	independiente
Instrucciones_02	true	true
Instrucciones_03	true	false
Instrucciones_04	true	independiente
Instrucciones_05	true	independiente

**Ejemplo 4.6:**

A continuación se plantea un problema para el cual se le pide construir un algoritmo que lo solucione, basándose en el uso de estructuras de selección anidadas.

Se necesita conocer cuál es el mayor de 3 números enteros *num1*, *num2* y *num3*, que ingresa un usuario al programa.  
¿Cómo se soluciona?

**Análisis:**

**Objetivo:** obtener el mayor de 3 números enteros.

**Entradas:** los 3 números que ingresa el usuario.

**Procesos:**

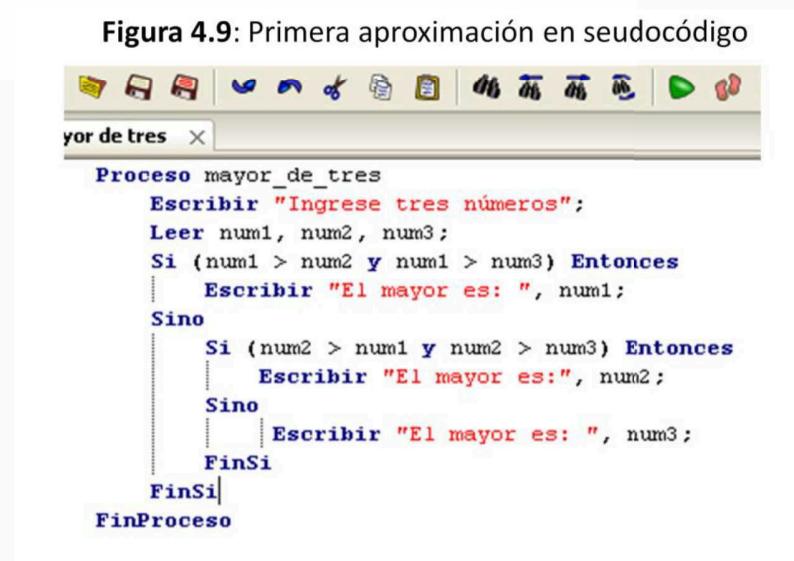
Deberemos comparar los números entre sí para saber cuál de ellos es el mayor de todos.

- Lectura de los 3 números
- La comparación de los 3 números y determinación de cuál de ellos es el mayor.

**Salida:** el mayor de los 3 números.

¿Existe una sola forma de solucionar este problema? ¿Cuántas formas crees tú que hay?

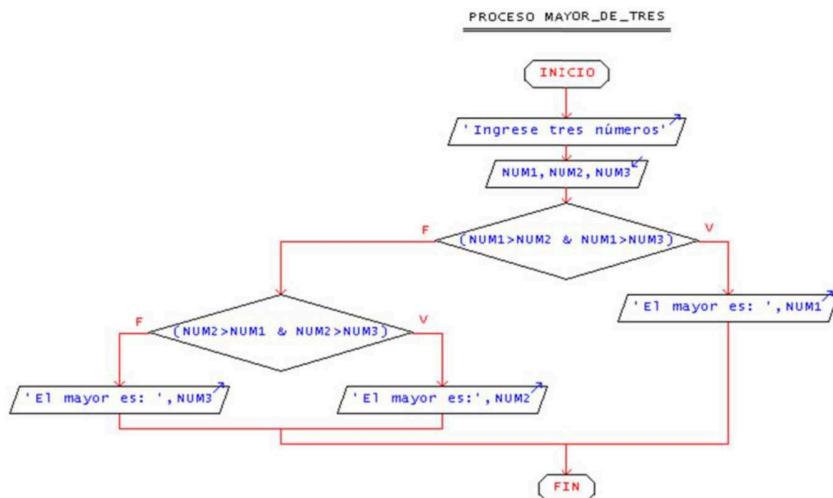
Aquí te presentamos tres aproximaciones diferentes para solucionar el mismo problema:

**Figura 4.9:** Primera aproximación en seudocódigo


```

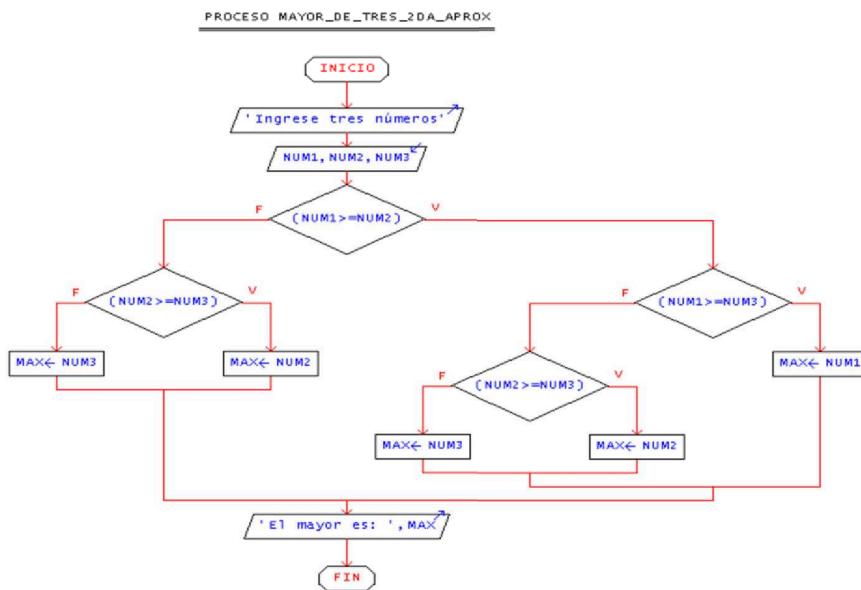
Proceso mayor_de_tres
    Escribir "Ingrese tres números";
    Leer num1, num2, num3;
    Si (num1 > num2 y num1 > num3) Entonces
        Escribir "El mayor es: ", num1;
    Sino
        Si (num2 > num1 y num2 > num3) Entonces
            Escribir "El mayor es:", num2;
        Sino
            Escribir "El mayor es: ", num3;
        FinSi
    FinSi
FinProceso

```

**Figura 4.10:** Primera aproximación en diagrama de flujoUna 2<sup>a</sup> aproximación:**Figura 4.11:** Segunda aproximación en seudocódigo

```

mayor de tres 2da aprox x
1  Proceso mayor_de_tres_2da_aprox
2      Escribir "Ingrese tres números";
3      Leer num1, num2, num3;
4      Si (num1 >= num2) Entonces
5          Si (num1 >= num3) Entonces
6              max <- num1;
7          Sino
8              Si (num2 >= num3) Entonces
9                  max <- num2;
10             Sino
11                 max <- num3;
12             FinSi
13         FinSi
14     Sino
15         Si (num2>= num3) Entonces
16             max <- num2;
17         Sino
18             max <- num3;
19         FinSi
20     FinSi
21     Escribir "El mayor es: ", max;
22
23 FinProceso
24
    
```

**Figura 4.12:** Segunda aproximación en diagrama de flujo

La 3º aproximación:

**Figura 4.13:** Tercera aproximación en seudocódigo y diagrama de flujo

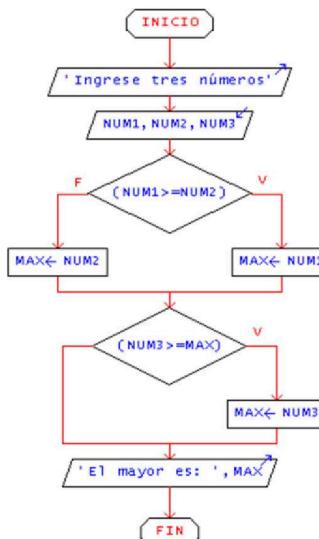
menú de la izquierda:

**mayor de tres 3da aprox**

```

1 Proceso mayor_de_tres_3ra_aprox
2   Escribir "Ingrese tres números";
3   Leer num1, num2, num3;
4   Si (num1 >= num2) Entonces
5     max <- num1;
6   Sino
7     max <- num2;
8   FinSi
9   Si (num3 >= max) Entonces
10    max <- num3;
11  FinSi
12  Escribir "El mayor es: ", max;
13 FinProceso
14
    
```

PROCESO\_MAYOR\_DE\_TRES\_3RA\_APROX



Analiza detalladamente cada solución y coméntala con tus compañeros.

Luego de haber discutido c/u de las soluciones mostradas traten de responder la siguiente pregunta:

¿Con qué solución nos quedamos?

Supongamos que los *valores de num1, num2 y num3* son distintos.

Revisemos la siguiente tabla que resume cada una de las aproximaciones mostradas, clasificadas según el número de condiciones que evalúa y cuántas asignaciones de variables realiza.

**Tabla 4.2:** Resumen de aproximaciones

Solución	Condiciones que evalúa	Asignaciones realizadas
1 <sup>a</sup> aproximación	siempre evalúa 2 condiciones	1
2 <sup>a</sup> aproximación	evalúa 2 ó 3 condiciones	1
3 <sup>a</sup> aproximación	siempre evalúa 2 condiciones	1 ó 2

Al analizar la tabla vemos que:

- la 1<sup>a</sup> aproximación siempre evalúa 2 condiciones y realiza una acción, por lo que podemos decir que es la más eficiente.
- la 2<sup>a</sup> aproximación evalúa 2 ó 3 condiciones y realiza una asignación, por lo cual podemos decir que es la menos eficiente.
- la 3<sup>a</sup> aproximación igualmente evalúa 2 condiciones, pero dependiendo de los *valores de num1, num2 y num3*, deberá realizar 1 ó 2 asignaciones. De la forma en la cual compara los valores, podríamos decir que es la más “elegante”.

Entonces tenemos 3 soluciones correctas, la pregunta ahora sería:

¿Cuál de las 3 soluciones elegimos?

La respuesta no es única. Siempre habrá que llegar a un compromiso entre eficiencia y elegancia (legibilidad, lo conciso del código, rapidez de desarrollo, reutilización, etc.).

**OJO!!!** Es importante destacar que independiente del tipo de bloque Si que se esté usando, siempre se ejecutará sólo una de las opciones presentes en el bloque. Luego el programa continuará ejecutándose en la instrucción que se encuentre inmediatamente a continuación del final del bloque Si.

#### 4.4 Estructuras de iteración

Las estructuras de control mostradas en la sección 4.3 funcionan muy bien en el caso de que una expresión deba evaluarse una sola vez, sin embargo, cuando se necesita que esa expresión sea evaluada muchas veces, resulta poco práctico. Por ejemplo, si se tienen que comparar las edades de todos los integrantes de un grupo de personas, se necesitaría construir tantos Si o Si/Sino, como integrantes hay en el grupo, además de tener una variable “edad”, por cada persona. Esto aumenta considerablemente la cantidad de líneas del programa y por ende la eficiencia del mismo (aparte de que se transforma en un programa muy tedioso de construir).

Una posible solución utilizando la estructura Si-Sino sería la siguiente:

Si (edad integrante 1 es mayor que edad integrante 2) entonces

    MayorEdad ← edad integrante 1

Sino

    Si (edad integrante 3 es mayor que MayorEdad) entonces

        MayorEdad ← edad integrante 3

    Sino

        Si (edad integrante 4 es mayor que MayorEdad) entonces

            MayorEdad ← edad integrante 4

        Sino

.....

.....

Si (edad integrante n es mayor que MayorEdad) entonces

    MayorEdad ← edad integrante n

    Fin Si

.....

.....

    Fin Si

    Fin Si

Fin Si

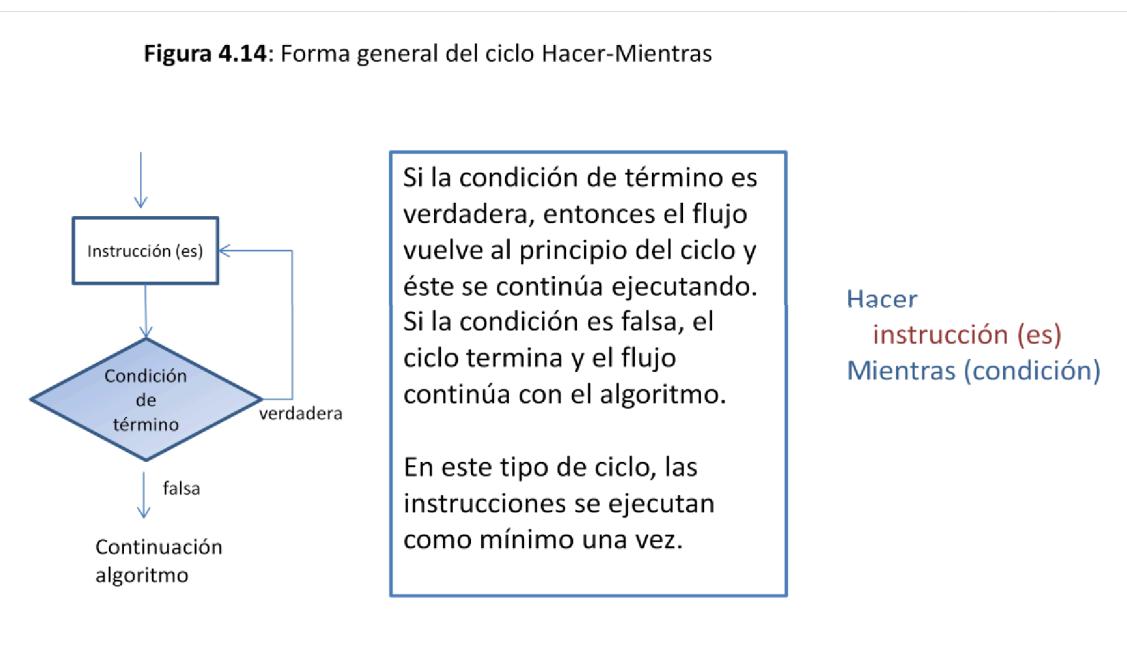
Para solucionar este tipo de situaciones, se utilizan estructuras denominadas de Iteración, las cuales pueden repetir un conjunto de instrucciones, una determinada cantidad de veces, sin tener que escribirlas cada vez.

Las estructuras de iteración o repetitivas son también conocidas como bucles, ciclos o lazos. Una estructura repetitiva permite la ejecución de una secuencia de sentencias una cierta cantidad de veces. Esto dependerá del tipo de solución que se deba implementar.

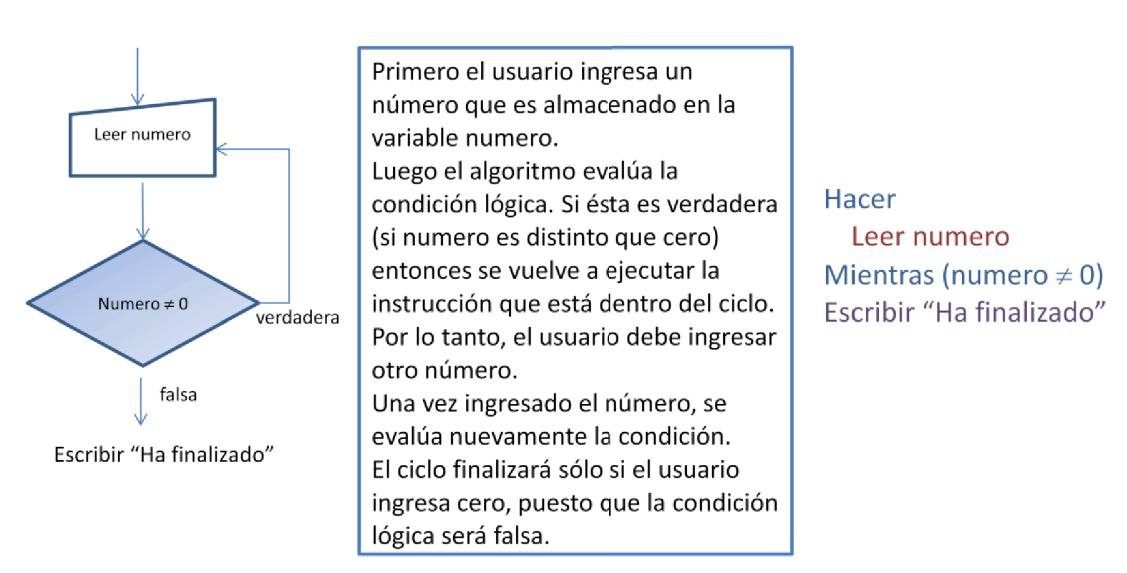
#### 4.4.1 Estructura iterativa Hacer-Mientras

Esta estructura repite el conjunto de instrucciones mientras la condición de término sea verdadera. La particularidad de este ciclo radica en que primero se ejecutan las sentencias y luego se evalúa la condición.

La figura siguiente nos muestra de manera general la estructura del ciclo en diagrama de flujo y seudocódigo:

**Figura 4.14:** Forma general del ciclo Hacer-Mientras**Ejemplo 4.7:**

El siguiente algoritmo permite a un usuario ingresar números. La entrada de datos finalizará cuando el usuario ingrese el número cero.

**Figura 4.15:** Ejemplo de uso del ciclo Hacer-Mientras

En este caso nos encontramos con un “**ciclo controlado por un centinela**”, ya que la ejecución del ciclo sólo finalizará cuando la variable que está en la condición lógica almacene un valor determinado. ¿Cuántas veces se ejecuta este ciclo?, ¿es posible determinarlo?, ¿cómo?.

Cuando se conoce exactamente la cantidad de veces que se deben ejecutar ciertas instrucciones, se puede utilizar un “**ciclo controlado por un contador**”.

Un **contador** es una variable donde se registra la cantidad de iteraciones que ejecuta un bucle. Estos pueden incrementarse o decrementarse en la unidad. Por lo general, los contadores se utilizan para evaluar el término de un ciclo.

La forma general de un contador es:

**Contador  $\leftarrow$  Contador +/- Constante**

Donde **Contador**, es la variable que permitirá contar la cantidad de veces que se ejecuta el bucle. Por ejemplo desde 1 hasta 10. Y **Constante**, es el incremento o decremento de la variable **Contador**. Así, si tenemos la constante 1, el contador irá aumentado de uno en uno cada vez que se ejecuta el bucle.

Para comprender mejor el uso de un contador, la figura siguiente muestra el estado de la variable **Contador** cuando esta se ejecuta varias veces.

**Figura 4.16:** Estado de la variable Contador cuando la cte es 1

Instrucciones	Zona de memoria	
Contador $\leftarrow$ 0	0	A la variable Contador se le asigna el valor (dato) cero
Contador $\leftarrow$ Contador + 1	1	A la variable Contador se le asigna el valor de Contador + 1 Como Contador actualmente contiene el valor cero, entonces sería 0+1 que es 1
Contador $\leftarrow$ Contador + 1	2	A la variable Contador se le asigna el valor de Contador + 1 Como Contador actualmente contiene el valor uno, entonces sería 1+1 que es 2
Contador $\leftarrow$ Contador + 1	3	

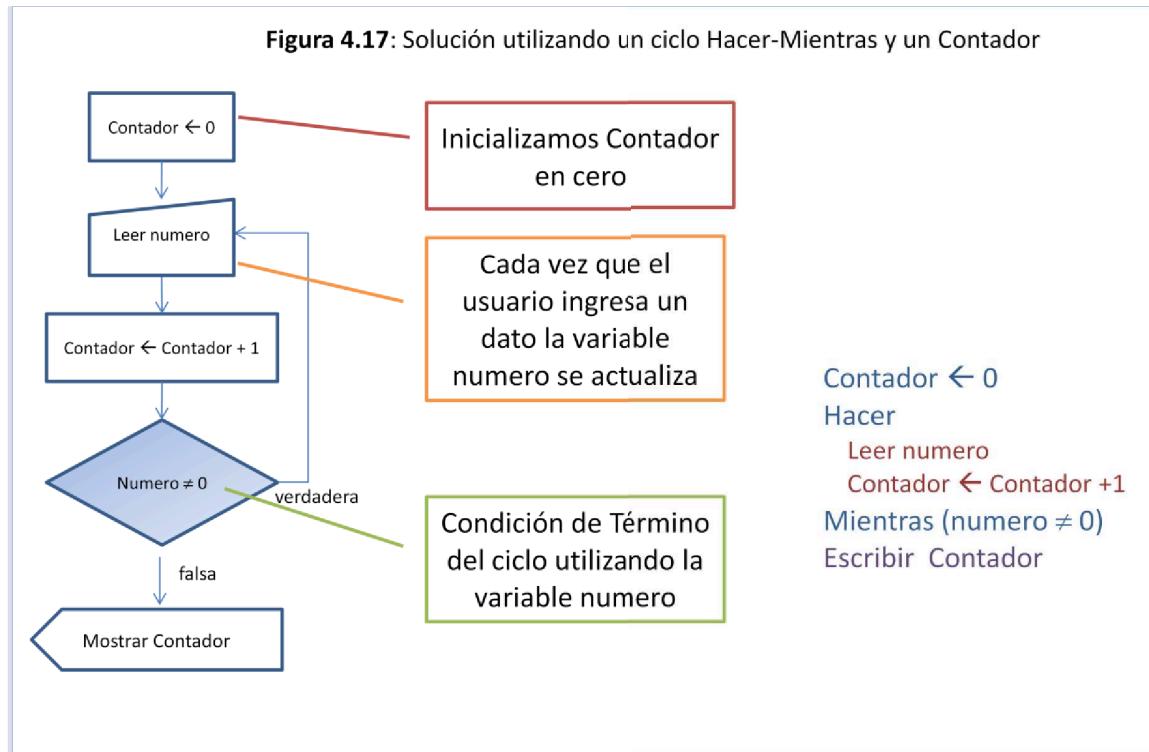
Además del contador, debemos conocer algunas características que debe poseer todo bucle o ciclo. Estos son al menos tres elementos:

- **Inicialización:** se refiere a establecer el punto de inicio de un ciclo, es decir marca el comienzo del ciclo. En la práctica se traduce a inicializar el valor de un variable, en un valor de partida.
  - Importante: por lo general una variable se inicializa en cero, pero no debe considerarse esto como una regla, la inicialización puede ser en cualquier valor, lo importante es que se regule adecuadamente la cantidad de iteraciones del ciclo.
- **Actualización:** se refiere a la actualización o modificación del valor de la variable que controla las iteraciones del ciclo (por lo general el contador). Si un contador no se actualiza, se produce un ciclo infinito.
- **Condición de Término:** La condición de término de un ciclo, determina hasta cuando se debe ejecutar el ciclo, es decir, marca el final del ciclo.

#### Ejemplo 4.8:

Ahora modificaremos el algoritmo del ejemplo 4.7 de tal manera de conocer la cantidad de números que ha ingresado el usuario.

La siguiente figura presenta una solución.



Observa que el **Contador** lo hemos inicializado en cero. Esto para que al actualizar la variable **Contador** ésta disponga de un valor almacenado que pueda ser sumado con el número 1 de la cte (Contador + 1, sería 0 + 1). De otro modo, causaría un error.

Observa también que la variable **numero** adquiere un valor diferente cada vez que se ejecuta el ciclo (la actualizamos), ya que el usuario puede ingresar cualquier número que desee. Esto es necesario, ya que es la variable que controla el ciclo (en la condición lógica) y en algún momento la condición lógica debe permitir la salida del mismo.

Una vez que finaliza el ciclo, podemos mostrar por pantalla el valor que almacena **Contador**, que es justamente “**la cantidad de veces que se ha ejecutado el ciclo**”.

Un ejemplo de ejecución de instrucciones paso a paso sería el siguiente:

Contador $\leftarrow 0$	
Leer numero	→ supongamos que el usuario ingresa el 6
Contador $\leftarrow$ Contador + 1	→ ahora almacena 1 ya que $0 + 1$ es 1
Se evalúa condición lógica (numero $\neq 0$ )	→ verdadero, ya que numero almacena el 6
Leer numero	→ supongamos que el usuario ingresa el 10
Contador $\leftarrow$ Contador + 1	→ ahora almacena 2 ya que $1 + 1$ es 2
Se evalúa condición lógica (numero $\neq 0$ )	→ verdadero, ya que numero almacena el 10
Leer numero	→ supongamos que el usuario ingresa el 3
Contador $\leftarrow$ Contador + 1	→ ahora almacena 3 ya que $2 + 1$ es 3
Se evalúa condición lógica (numero $\neq 0$ )	→ verdadero, ya que numero almacena el 3
Leer numero	→ supongamos que el usuario ingresa el 0
Contador $\leftarrow$ Contador + 1	→ ahora almacena 4 ya que $3 + 1$ es 4
Se evalúa condición lógica (numero $\neq 0$ )	→ falso, ya que numero almacena el 0
Muestra 4 por pantalla	

La tabla siguiente muestra el estado de las variables de acuerdo al ejemplo de ejecución anterior.

Tabla 4.3: Estado de las variables

Operador	Descripción
$0$	
$0 + 1 = 1$	6
$1 + 1 = 2$	10
$2 + 1 = 3$	3
$3 + 1 = 4$	0

Ahora haré varias preguntas que puedes responder junto a tus compañeros: ¿Cuál es la utilidad de un contador?, ¿En qué casos podríamos utilizar una cte. distinta a uno?

Ahora supongamos que deseamos modificar el algoritmo del ejemplo 4.8 para calcular el promedio de los números que ha ingresado el usuario.

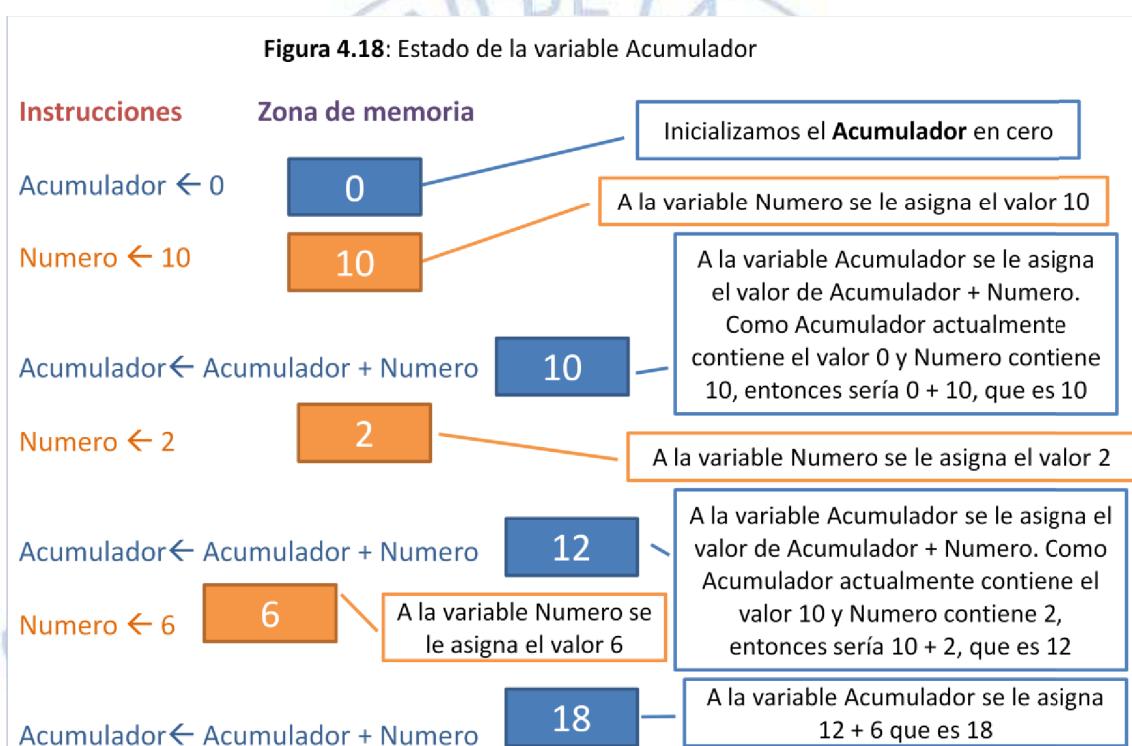
En el ejemplo anterior el usuario ha ingresado varios números (por ejemplo, 6, 10, 3 y 0), sin embargo la variable **numero** tendrá sólo un valor al finalizar el ciclo, el cero. De esta manera no podemos sumar los números, por lo que necesitaremos una variable especial, conocida como **acumulador o sumador**.

Un **acumulador** es una variable que incrementa su valor, en una cantidad x (distinta de la unidad), por cada iteración. Estos cantidades x pueden ser ingresados por el usuario, obtenidos desde algún archivo o

generados de manera aleatoria. La forma general de un **acumulador** es la siguiente:

Acumulador  $\leftarrow$  Acumulador + Variable

Para comprender mejor el uso de un **acumulador**, la siguiente figura muestra el estado de la variable **acumulador** cuando esta se ejecuta varias veces.



La tabla siguiente muestra el estado de las variables de acuerdo a la ejecución de las instrucciones de la figura anterior.

**Tabla 4.4: Estado de las variables**

Acumulador	Numero
0	10
10	2
12	6
18	-

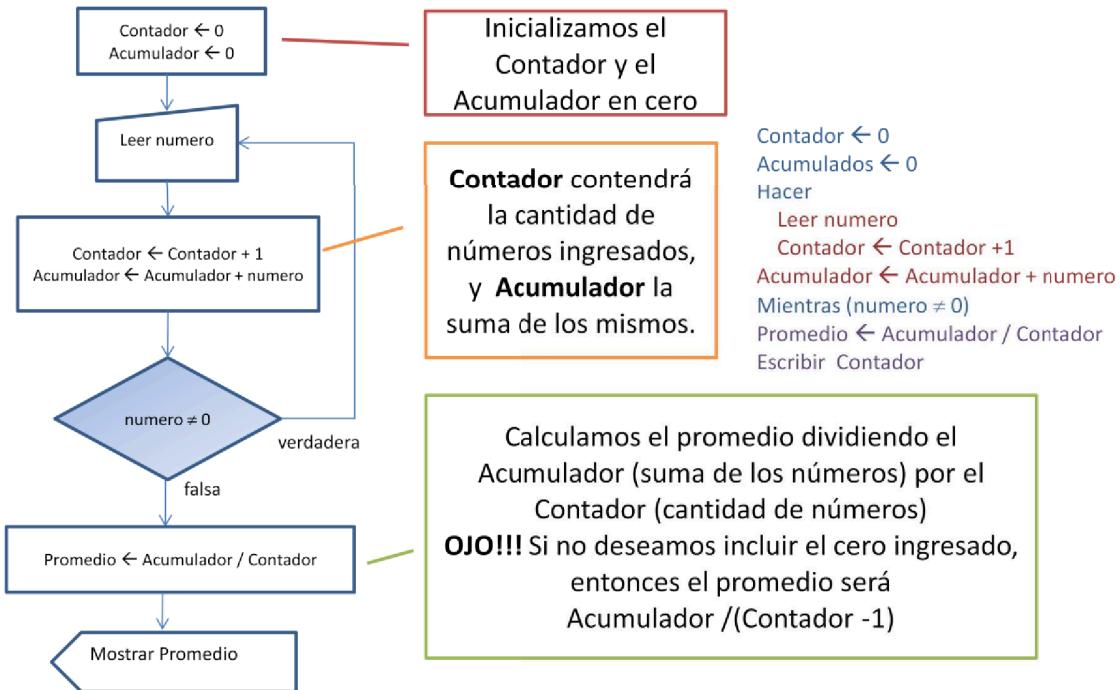
Observamos que el finalmente el **Acumulador** almacena el valor 18, que es la suma de  $10 + 2 + 6$ , que son los números que se le ha asignado a la variable **Número**.

#### Ejemplo 4.9:

En este ejemplo modificaremos el algoritmo del ejemplo 4.8 para calcular el promedio de los números que ha ingresado el usuario.

La siguiente figura presenta una solución.

**Figura 4.19:** Solución utilizando un ciclo Hacer-Mientras y un Acumulador



Observa que tanto el **Contador** como el **Acumulador** lo debemos inicializar en cero, ya que en ese momento no se ha ingresado números y por tanto tampoco se ha sumado alguno. Una vez que el usuario ingresa un número, debemos contarla y además sumarla. Es por ello que disponemos de dos instrucciones muy importantes, y que deben estar ubicadas dentro del ciclo.

**Contador ← Contador + 1**  
un número con la instrucción Leer numero

Que será la variable que cuente de uno en uno (cada vez que se ingresa

**Acumulador ← Acumulador + numero**

Que será la variable que sume los números ingresados

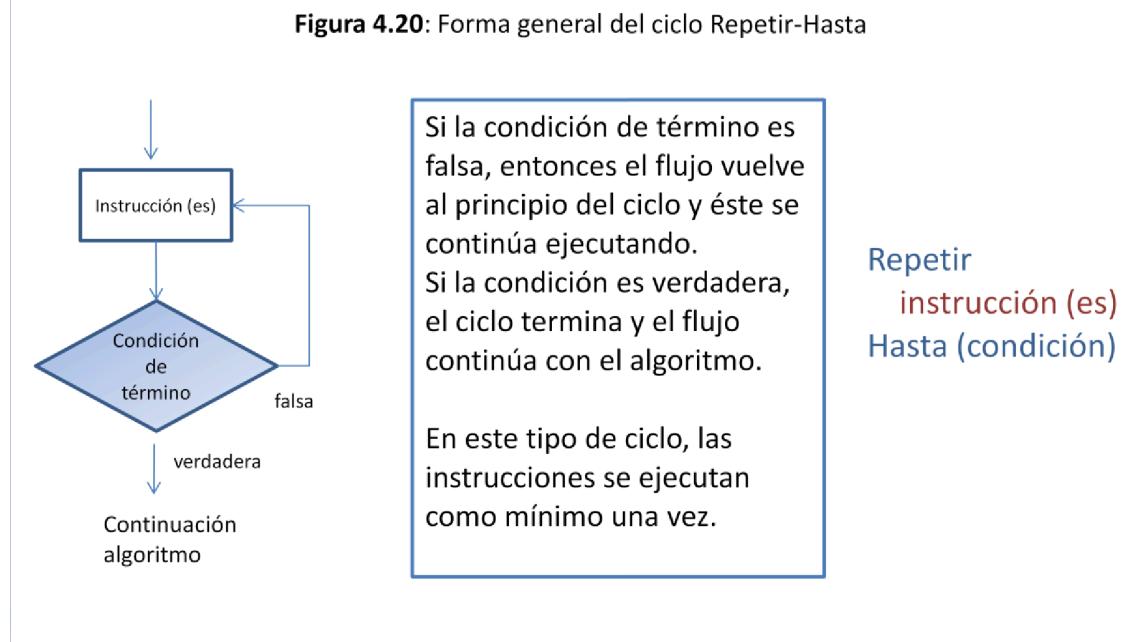
También observamos que para calcular el promedio necesitamos conocer la suma de los números ingresados por el usuario y la cantidad. Es por ello, que dividimos Acumulador / Contador y almacenamos este resultado en otra variable, **Promedio**.

Una variante de este mismo tipo de bucle es el ciclo Repetir-Hasta que veremos en la siguiente sección.

#### 4.4.2 Estructura iterativa Repetir-Hasta

Esta estructura repite un conjunto de instrucciones hasta que la condición lógica resulta ser verdadera. Por lo tanto, las instrucciones se volverán a ejecutar sólo si la condición es falsa. La figura siguiente muestra la forma general de este tipo de ciclo.

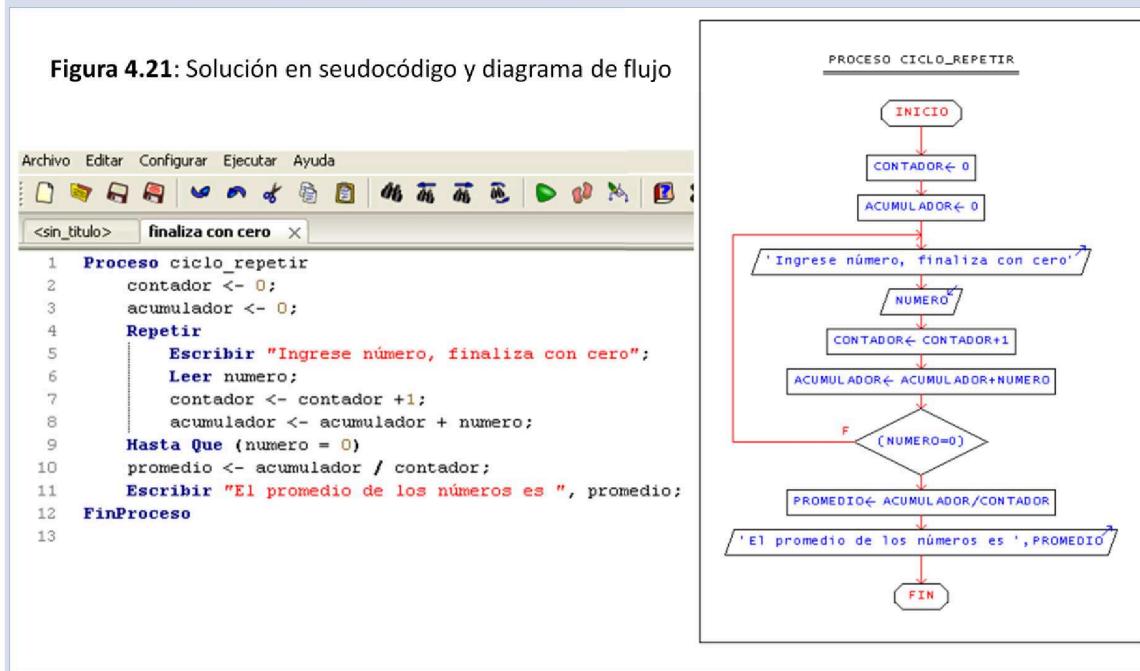
Figura 4.20: Forma general del ciclo Repetir-Hasta



### Ejemplo 4.10:

En este ejemplo te mostramos la solución en PSeInt para el problema presentado en el ejemplo 4.9.

**Figura 4.21:** Solución en seudocódigo y diagrama de flujo

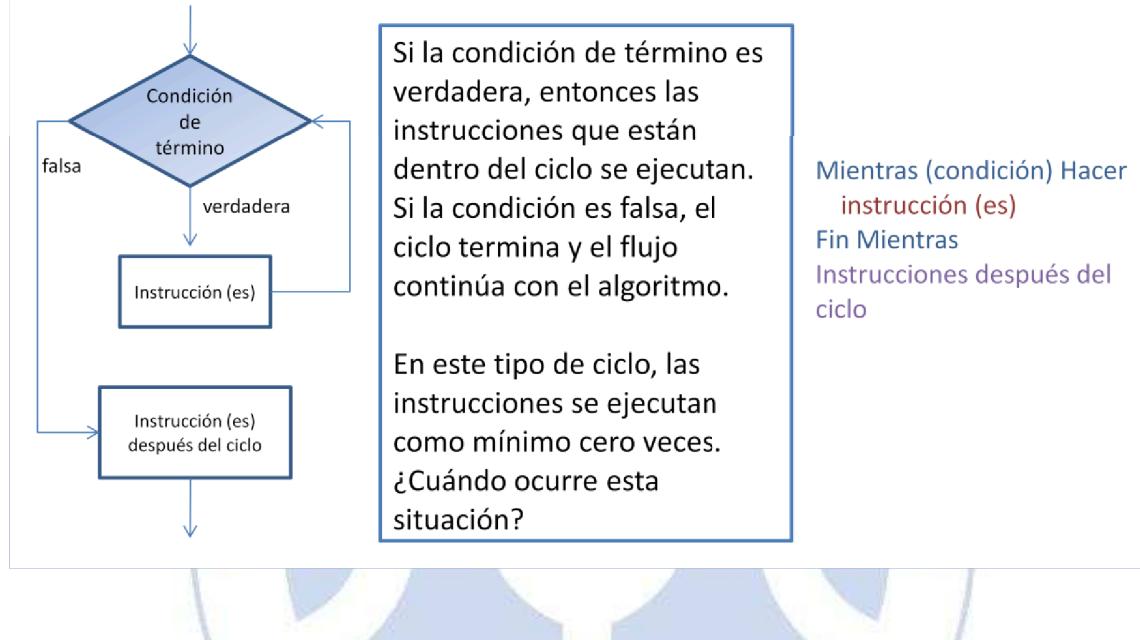


#### 4.4.3 Estructura iterativa Mientras-Hacer

Esta estructura repite el conjunto de instrucciones que se encuentra dentro del ciclo, mientras la condición evaluada sea verdadera. Si la condición es falsa, entonces se termina el ciclo. En este caso, primero se evalúa la condición y luego se ejecutan las acciones que se encuentran dentro del ciclo, si correspondiese.

De manera gráfica tenemos:

**Figura 4.22:** Forma general del ciclo Mientras-Hacer

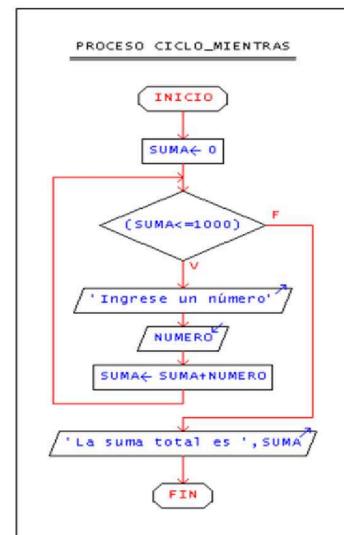


#### Ejemplo 4.11:

Supongamos que se requiere un algoritmo que sume los números ingresados por el usuario hasta que la suma sea mayor que 1000. Una posible solución en la herramienta PSeInt se muestra en la figura siguiente.

**Figura 4.23:** Solución en seudocódigo y diagrama de flujo

```
finaliza por tarea x
1  Proceso ciclo_mientras
2      suma <- 0;
3      Mientras (suma <= 1000) Hacer
4          Escribir "Ingrese un número";
5          Leer numero;
6          suma <- suma + numero;
7      FinMientras
8      Escribir "La suma total es ", suma;
9  FinProceso
10
```



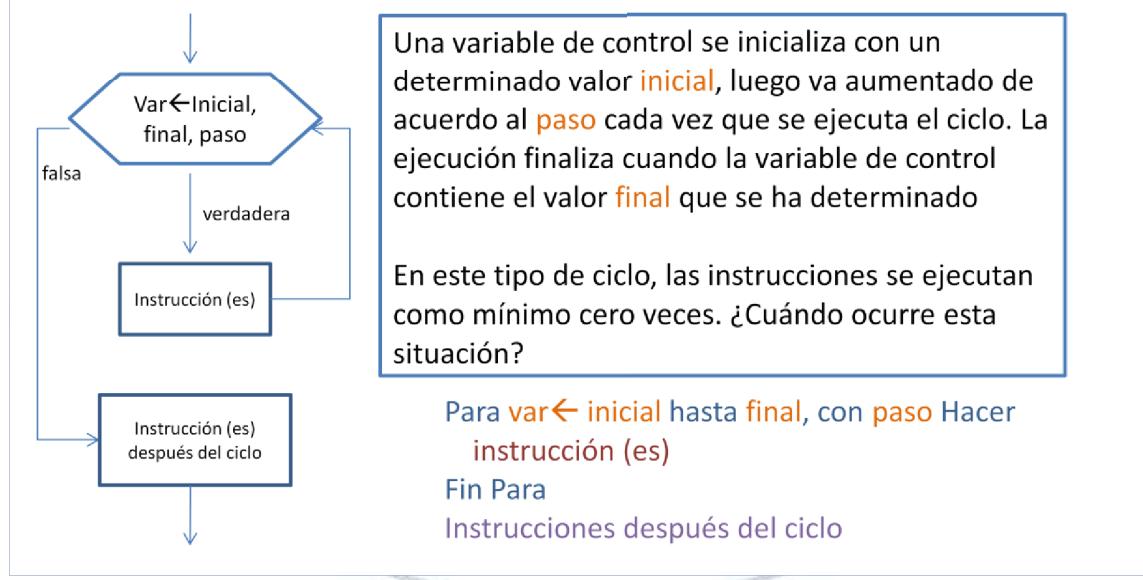
Observamos que la variable **suma** es inicializada en cero para no tener problemas cuando se ejecute la primera suma ( $\text{suma} \leftarrow \text{suma} + \text{numero}$ ). Cuando el ciclo **Mientras-Hacer** se ejecuta, lo primero que hace es evaluar la condición lógica. Como esta es verdadera (es decir, **suma** es menor o igual que 1000), entonces ejecuta las tres instrucciones que están ubicadas dentro del ciclo. Una vez que se ejecutan las instrucciones, se vuelve a evaluar la condición lógica. Que sea verdadera o falsa dependerá del número que ha ingresado el usuario. Si la **suma** sigue siendo menor o igual a 1000, entonces las tres instrucciones que están dentro del ciclo se volverán a ejecutar. El ciclo sólo finalizará si **suma** es mayor que 1000. En tal caso se ejecutará la instrucción que está después del ciclo (Escribir "la suma total es ", **suma**).

A este tipo de ciclo, se le llama comúnmente "**ciclo controlado por un tarea**", ya que la ejecución del ciclo sólo finalizará si se completa la tarea solicitada, que en este caso fue sumar números hasta que la **suma** sea mayor que 1000.

#### 4.4.4 Estructura iterativa Para

Este tipo de ciclo repite las instrucciones una cantidad determinada de veces. Esta cantidad está dada por un valor de finalización de la variable de control del ciclo y el valor del paso o incremento. La figura siguiente presenta la estructura general en diagrama de flujo y seudocódigo.

**Figura 4.24:** Forma general del ciclo Mientras-Hacer

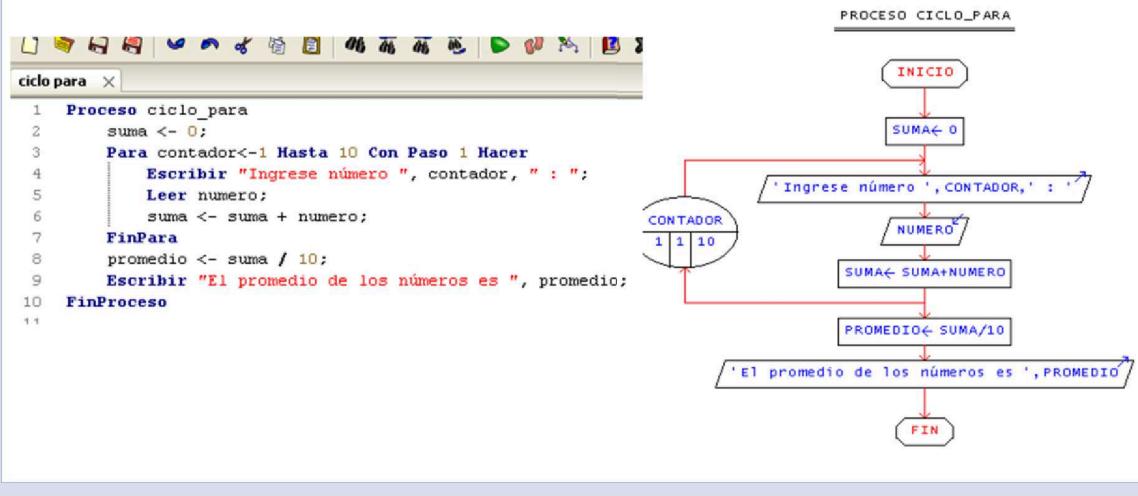


Describiremos la ejecución de este ciclo con un ejemplo.

#### Ejemplo 4.12:

El siguiente algoritmo calcula el promedio de 10 números ingresados por el usuario.

Figura 4.25: Solución en seudocódigo y diagrama de flujo



El ciclo para está controlado por una variable que es el **contador**. Este contador se inicializa en 1 que es valor inicial que se ha determinado y finaliza en 10 que es valor final. Además el contador irá aumentando de uno en uno cada vez que se ejecuten las instrucciones del ciclo. Por lo tanto el ciclo finalizará cuando se hayan ejecutado exactamente 10 veces dichas instrucciones.

La siguiente tabla presenta un ejemplo del estado de las variables al ejecutar el algoritmo. Supondremos que el usuario ingresa los números que se observan en la tabla.

Tabla 4.5: Estado de las variables

contador	numero	suma
-	-	0
1	2	2
2	1	3
3	3	6
4	50	56
5	-6	50
6	2	52
7	6	58
8	-6	52
9	4	56
10	1	57

¿Cuál sería la solución utilizando el ciclo Mientras-Hacer?. Explique las diferencias.

## 4.5 Ejercicios resueltos

**Ejemplo 4.13:** ¿Son equivalentes las siguientes estructuras de control? Sí/No. Justifique su respuesta.

Primer caso. SI...SINO	Segundo Caso: SI...SINO anidados
<pre>SI (c1 Y c2) ENTONCES     bloque_A SINO     bloque_B FIN SI</pre>	<pre>SI (c1) ENTONCES     SI (c2) ENTONCES         bloque_A     SINO         bloque_B     FIN SI SINO     bloque_B FIN SI</pre>

Revisemos estas estructuras de control con más detalle.

### Primer Caso: Solución basada en SI...SINO

bloque\_A se ejecuta cuando: c1 y c2 son verdaderas

bloque\_B se ejecuta cuando: (c1 y c2) es falso, esto puede ser por qué:

- c1 es verdadera y c2 es falsa
- c1 es falsa y c2 es verdadera
- c1 es falsa y c2 es falsa

### Segundo Caso: Solución basada en SI...SINO anidados

bloque\_A se ejecuta cuando: c1 y c2 son verdaderas

bloque\_B se ejecuta cuando: c1 es verdadera y c2 es falsa o bien cuando c1 es falsa

### Análisis de las soluciones

Si revisas con atención ambos casos y verificas cuando se ejecuta cada bloque de instrucciones para el caso desarrollado con SI...SINO y SI...SINO anidados, podrás darte cuenta que no son equivalentes. ¿Te queda claro por qué?

Luego si ya estás convencido de que ambos casos no son equivalentes, es decir, no dan el mismo resultado, surge otra pregunta: ¿Cuál te parece menos compleja?

**Ejemplo 4.14:** El problema a resolver es calcular el área y perímetro de un triángulo rectángulo. Este ejercicio corresponde al ejemplo 3.2 del capítulo 3, sólo que ahora utilizaremos las estructuras de control revisadas en este capítulo para resolver el problema de manera más completa.

**Análisis:**

Para poder resolver el problema necesitamos conocer las características de un triángulo rectángulo. Ver figura 3.2 del capítulo anterior.

Para calcular el perímetro del triángulo rectángulo se sabe que la fórmula es:

$$(1) \text{perímetroT} = \text{catetoBase} + \text{catetoAltura} + \text{hipotenusa}$$

Se sabe además que los catetos y la hipotenusa no pueden tener cualquier valor, ya que se debe cumplir la siguiente relación.

$$(2) \text{catetoBase}^2 + \text{catetoAltura}^2 = \text{hipotenusa}^2$$

Con esta descripción del problema en el capítulo 3 se determinaron las entradas y salidas de la solución.

**Entradas:** ambos catetos (variables `catetoBase`, `catetoAltura`).

**Salidas:** perímetro del triángulo rectángulo (variable `Perímetro`)

Además se debe incorporar una restricción a los datos de entrada: estos no pueden ser números negativos, ya que se ingresa el valor de medida de los catetos. Para ello debemos enviarle un mensaje al usuario avisando esta restricción, y en caso de equivocarse entregarle algún mensaje de error.

**Diseño:**

Proceso: Primero se debe calcular la hipotenusa una vez que se ingrese de manera correcta el valor se cada cateto. El siguiente paso es calcular el perímetro del triángulo de acuerdo a la ecuación (1).

Una primera aproximación para solucionar el problema es:

**Ingresar valores positivos para los catetos**

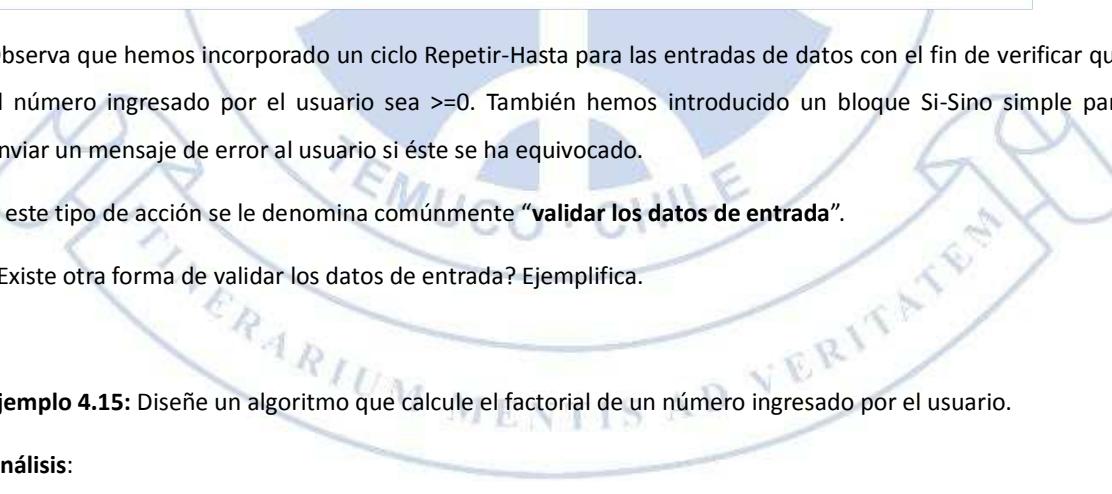
**Calcular hipotenusa**

**Calcular perímetro**

**Mostrar perímetro**

La siguiente figura presenta una posible solución al problema utilizando PseInt.

**Figura 4.26:** Solución en seudocódigo



```

perímetro triangulo con validacion x
1 Proceso calcular_perímetro
2   Repetir
3     Escribir "Ingrese un número positivo para el primer cateto";
4     Leer catetoA;
5     Si (catetoA <0) Entonces
6       Escribir "Error, debe ingresar un número positivo";
7       FinSi
8     Hasta Que (catetoA>=0)
9     Repetir
10     Escribir "Ingrese un número positivo para el segundo cateto";
11     Leer catetoB;
12     Si (catetoB <0) Entonces
13       Escribir "Error, debe ingresar un número positivo";
14       FinSi
15     Hasta Que (catetoB>=0)
16     hipotenusa <- RC (catetoA*catetoA + catetoB*catetoB);
17     perímetro <- catetoA + catetoB + hipotenusa;
18     Escribir "El perímetro del triángulo es ", perímetro;
19 FinProceso
20

```

Observa que hemos incorporado un ciclo Repetir-Hasta para las entradas de datos con el fin de verificar que el número ingresado por el usuario sea  $\geq 0$ . También hemos introducido un bloque Si-Sino simple para enviar un mensaje de error al usuario si éste se ha equivocado.

A este tipo de acción se le denomina comúnmente “**validar los datos de entrada**”.

¿Existe otra forma de validar los datos de entrada? Ejemplifica.

**Ejemplo 4.15:** Diseñe un algoritmo que calcule el factorial de un número ingresado por el usuario.

**Análisis:**

En este paso se deben determinar las entradas y salidas que requiere el problema. Se pide calcular el factorial de un número ingresado, por tanto la entrada es un solo número, y la salida es el factorial de éste. Además se sabe que:

$$(1) \quad 0! = 1$$

$$(2) \quad N! = (N-1) * (N-2) * (N-3) * \dots * 1$$

Donde N es un número  $\geq 0$

Por lo tanto, el número N ingresado por el usuario debe ser  $\geq 0$  (nuevamente debemos validar la entrada).

**Entradas:** Un número mayor o igual que cero (variable n)

**Salida:** El factorial del número n

#### Diseño:

Proceso: Para calcular el factorial de un número, necesitamos realizar un producto acumulativo de los números sucesivos desde el 1 hasta el número ( $1 * 2 * 3 * \dots * N$ ), por lo que necesitaremos dos variables: una que genere los números desde el 1 hasta el número ingresado, y la otra que acumule el producto de los mismos.

Para generar números sucesivos podemos utilizar una variable del tipo contador y un ciclo Mientras-Hacer.

Para generar un producto acumulado, necesitamos una variable del tipo acumulador. Sólo que multiplicaremos los valores que va generando Contador en vez de sumarlos.

Una primera aproximación para solucionar el problema es:

**Ingresar un número para N (mayor o igual que cero)**

**Iniciar acumulador en 1**

**Iniciar contador en 1**

**Iniciar ciclo (finaliza cuando contador sea mayor que N)**

**Actualizar acumulador multiplicando el acumulador actual con el contador**

**Aumentar contador en uno**

**Fin ciclo**

**Mostrar factorial por pantalla**

Una solución en seudocódigo sería la siguiente:

**Repetir**

**Escribir “Ingresé un número mayor o igual que cero”**

**Leer N**

```

Hasta (N >=0)
Acumulador ← 1
Contador ← 1
Mientras (Contador <= N) Hacer
    Acumulador ← Acumulador * Contador
    Contador ← Contador + 1
Fin Mientras
Escribir "El factorial de ", N , "es: ", Acumulador
Fin algoritmo

```

La siguiente tabla muestra el estado de las variables cuando el usuario ingresa el número 4.

**Tabla 4.6:** Estado de las variables

N	Contador	Acumulador
4	1	1
4	2	1 * 2 = 2
4	3	2 * 3 = 6
4	4	6 * 4 = 24

Por lo tanto, la salida del algoritmo será el número 24 que justamente es el factorial de 4!

**Ejemplo 4.16:** Se requiere un algoritmo para calcular el resultado de la siguiente función F(x)



**Análisis:**

Necesitamos definir las entradas y salidas del problema que se presenta. Observamos que necesitamos ingresar un solo número ( $x$ ), y que la salida presenta tres posibilidades: cuando  $x > 0$ , cuando  $x = 0$  y cuando  $x < 0$ .

**Entradas:** Un número cualquiera (variable x)

**Salida:** Tenemos tres posibles salidas: el resultado de  $x!$ , cero, o el resultado de la sumatoria de  $i$  desde  $x$  hasta  $-1$

**Diseño:**

Proceso: En el ejemplo 4.15 ya mostramos cómo calcular el factorial de un número, y en el ejemplo 4.9 mostramos cómo se acumula. Pero como tenemos tres posibles salidas, necesitaremos una estructura de control Si-Sino para determinar dicha salida de acuerdo al valor de x ingresado.

Una primera aproximación sería la siguiente:

Ingresar x

Si ( $x > 0$ )

Sino

Calcular  $x!$

Resultado  $\leftarrow$  Acumulador

Si ( $x = 0$ )

Resultado  $\leftarrow 0$

Sino

Calcular sumatoria de i desde x hasta cero

Resultado  $\leftarrow$  sumatoria

Fin Si

Fin Si

Mostrar El resultado de F(x)

Fin algoritmo

El seudocódigo completo será:

Escribir "Ingrese un número para x"

Leer x

Si ( $x > 0$ )

Acumulador  $\leftarrow 1$

Contador  $\leftarrow 1$

Mientras (Contador  $\leq x$ ) Hacer

    Acumulador  $\leftarrow$  Acumulador \* Contador

    Contador  $\leftarrow$  Contador + 1

Fin Mientras

Sino

    Resultado  $\leftarrow$  factorial

Si ( $x = 0$ )

    Resultado  $\leftarrow 0$

    Sino

        Acumulador  $\leftarrow 0$

        Contador  $\leftarrow x$

        Mientras (Contador  $\leq -1$ ) Hacer

            Acumulador  $\leftarrow$  Acumulador + Contador

            Contador  $\leftarrow$  Contador + 1

        Fin Mientras

    Resultado  $\leftarrow$  sumatoria

```
Fin Si  
Fin Si  
Escribir "F(x) = ", Resultado  
Fin algoritmo
```

**Ejemplo 4.17:** Diseñe un algoritmo en pseudocódigo o diagrama de flujo para resolver la siguiente serie:

$$(1) \quad e^x = 1 + x + x^2/2! + x^3/3! + x^4/4! + \dots + x^n/n!$$

Debe describir las **entradas** que se requieren, **proceso** y **salida**. En el proceso debe describir las estructuras de control que necesita para resolver el problema, además de los sumadores, contadores y variables.

#### Análisis

**Entradas:** Se requiere que el usuario ingrese dos datos, el de X y el de n.

**Salida:** Se debe mostrar el resultado de la serie, es decir, la suma total de cada término

#### Diseño:

Proceso: Como es una serie, se necesita una variable del tipo acumulador para sumar los términos de la serie. También se requiere una variable del tipo contador para generar números desde el 0 hasta n.

Para realizar la suma de los términos, se requiere dos ciclos, uno para calcular  $x^{\text{cont}}$  y  $\text{cont}!$ ; y el otro ciclo para sumar  $x^{\text{cont}}/\text{cont}!$ .

#### Solución en Pseudocódigo

Proceso calcula\_serie

Repetir

    Escribir "Ingrese un número >= 0 para n";

    Leer n;

    Hasta que (n>=0)

    Escribir "Ingrese un valor para X";

    Leer X

    Suma <-1;

    Fact <-1;

    cont <- 1;

    Mientras (cont <=n) Hacer

        Multi <-X;

        Para i<-2 hasta cont Con Paso 1 Hacer

```
Multi <- Multi *X;  
Finpara  
Fact <- Fact * cont;  
Suma <- Suma + Multi /Fact;  
cont <- cont +1;  
FinMientras  
Escribir "el valor de ex es ", Suma;  
FinProceso
```

## 4.6 Ejercicios propuestos

**Ejercicio 4.1:** Construya un algoritmo usando DFD y seudocódigo, que permita saber cuál es el mayor de 4 números que ingresa el usuario.

**Ejercicio 4.2:** Una importante entidad financiera te pide construir un seudocódigo que permita realizar un clasificador de dinero similar al que usan los cajeros automáticos, es decir dada una cantidad de dinero que se solicita, el algoritmo entrega un desglose a partir de las unidades monetarias existentes.

Ej.: Se ingresa la cantidad de \$21.511, la solución debe entregar:

1 billete de \$20.000

0 billetes de \$10.000

0 billetes de \$5.000

0 billetes de \$2.000

1 billete de \$1.000

1 moneda de \$500

0 monedas de \$100

0 monedas de \$50

1 moneda de \$10

0 monedas de \$5

1 moneda de \$1

**Ejercicio 4.3:** Construya un algoritmo que dé una solución al siguiente problema. Dada una NOTA que ingresa

el usuario entre 1.0 y 7.0, la transforme a una nota conceptual según la lista mostrada a continuación:

Si la nota está entre:

**1,0 y 3,9 → Insuficiente**

**4,0 y 4,9 → Suficiente**

**5,0 y 5,9 → Bien**

**6,0 y 7,0 → Muy bien**

**Ejercicio 4.4:** Escribir un algoritmo en diagrama de flujo y seudocódigo que permita calcular y mostrar por pantalla el sueldo de un empleado de una empresa. El sueldo es función de las horas trabajadas, el sueldo base y los descuentos. Se sabe que cuando un empleado trabaja más de 40 horas, la empresa pagará las horas extras de acuerdo a la siguiente tabla de valores:

Cantidad horas extras	Pago por hora (\$)
Entre 1 y 3	\$2.000
Entre 4 y 7	\$3.000
Más de 7	\$3.500

El sueldo base es asignado a cada empleado de acuerdo al grado que le entregue la empresa, como muestra la siguiente tabla:

Grado	Sueldo Base
1	\$450.000
2	\$550.000
3	\$700.000

Los descuentos que se realizan son por Salud y por el Fondo Previsional, que son distintos para cada trabajador según las empresas que contraten. Las siguientes tablas muestran las empresas de Salud (ISAPRES) y de Fondo Previsional (AFP) con los descuentos que realizan sobre el sueldo base:

ISAPRE	% descuento	AFP	% Descuento
1 Colmena	6,5%	1 BanSander	12%
2 Ban Médica	7,3%	2 Geometrica SA	13,4%
3 Vida 3	7%	3 Cumprum	12,3%
4 Fonasa	7%	4 Provida	12,9%

Para dar solución al problema es necesario que describa las **entradas, proceso y salidas**. Para describir el proceso, debe identificar las estructuras de control que necesite y variables.

**Ejercicio 4.5:** Complemente la solución lograda en el ejercicio 4.4 para que el algoritmo muestre el sueldo de los N empleados que tiene la empresa.

#### 4.7 Comentarios finales

El objetivo central de este capítulo fue mostrarte cómo desarrollar soluciones y algoritmos cuyas estructuras permitirán solucionar problemas de mayor complejidad que lo realizado hasta el capítulo anterior. Cuando los problemas presentan una mayor complejidad no es posible solucionarlo mediante un algoritmo que se ejecute secuencialmente, instrucción por instrucción. Es por ello que hemos descrito las estructuras básicas de control que permiten generar varios caminos alternativos para dichos algoritmos. Estas estructuras son: las selectivas y las iterativas.

Finalmente, hemos descrito una serie de ejemplos resueltos paso a paso con el fin de que comprendas de mejor manera la forma cómo se utiliza cada estructura de control.

#### 4.8 Referencias

Harvey M. Deitel, Paul J. Deitel, Pearson Educación (2004): “**Como programar en Java, 7<sup>a</sup> Edición**”. Editorial Pearson.

