



## Estructuras de control

### Objetivo

- Conocer los diferentes tipos de estructuras lógicas.
- Conocer y poner en práctica estructuras de tipo condicional.
- Conocer y poner en práctica estructuras de iteración.
- Conocer y poner en práctica estructuras de selección
- Conocer y poner en práctica operadores lógicos

### Contenidos

Estructuras algorítmicas básicas: condición, iteración, selección. Lógica formal.  
Variables tipo contador y variables tipo acumulador.

### Estructuras de control



En esta unidad nos proponemos explicarte las estructuras de control elementales de un programa. Te presentaremos estructuras condicionales para la toma de decisiones, estructuras iterativas para realizar una acción varias veces y estructuras de selección para realizar alguna acción de acuerdo al valor de una variable.

¿Comenzamos?

### Estructuras de control condicionales

En los algoritmos presentados previamente te has familiarizado con las instrucciones del lenguaje pseudocódigo para ingresar y mostrar datos y para asignar un valor a una variable.

Te proponemos un ejemplo:

#### SEUDOCÓDIGO PseInt

```
Algoritmo CalcularPromedio
  Definir NOTA1, NOTA2, NOTA3 Como Entero
  Definir PROMEDIO Como Real

  Escribir "Ingrese nota 1"
  Leer NOTA1
  Escribir "Ingrese nota 2"
  Leer NOTA2
  Escribir "Ingrese nota 3"
```

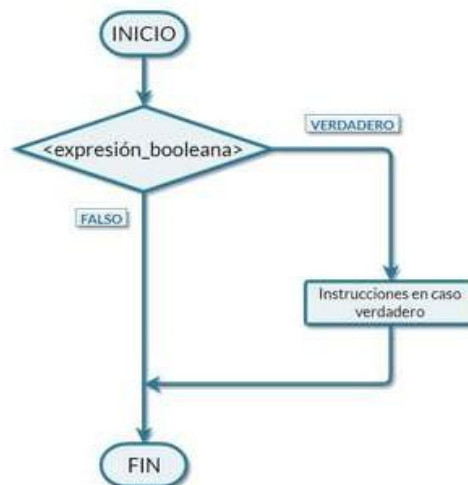
Leer NOTA3

```
PROMEDIO <- (NOTA1+NOTA2+NOTA3) / 3  
Escribir "El promedio de notas es:", PROMEDIO  
FinAlgoritmo
```

En todos los algoritmos que realizamos hasta ahora, como en este ejemplo, seguramente habrás observado que, sin importar los valores de los datos o alguna condición en especial, siempre se ejecutaban todas las instrucciones.

Ahora bien, como en la vida diaria, donde por ejemplo ante el resultado de un promedio hay que decir si se aprobó o no, en los algoritmos también nos encontramos con estas situaciones. Hay momentos donde, de acuerdo a una determinada situación, el programa deberá realizar alguna instrucción. Por lo tanto, tenemos la necesidad de contar con una herramienta que nos permita poder ejecutar o no, un grupo de acciones ante una situación. Para esto vamos a utilizar el **condicional**.

### Estructura condicional simple



En cierto momento, definido por el programador, la computadora evalúa una condición, es decir, una expresión booleana, que se representa mediante un rombo. Si la expresión devuelve un resultado VERDADERO, la computadora ejecuta las instrucciones dentro de un bloque especial que luego retorna al flujo original, de lo contrario, el flujo continúa normalmente.

La sintaxis en pseudocódigo es la siguiente:

**Si (<expresión\_booleana>) Entonces**

**<instrucciones>**



## FinSi

Si se tipea literalmente, indica que vamos a iniciar un bloque de selección.

<expresión\_booleana> es la condición que la computadora evaluará. El hecho de estar entre paréntesis es opcional, aunque te recomiendo que lo hagas para mejorar la legibilidad y porque muchos lenguajes formales requieren que la condición esté obligatoriamente encerrada entre paréntesis.

Entonces se tipea literalmente, indica que a continuación serán listadas las instrucciones que se ejecutarán en caso de que la condición sea VERDADERO.

<instrucciones> son las instrucciones que se ejecutarán. Puede ser solo una, o varias, siguiendo los conceptos vistos hasta aquí.

FinSi se tipea literalmente, indica que ha terminado el bloque de selección.

Un programa sencillo que facilita la comprensión podría ser el siguiente: pedirle al usuario que ingrese su edad. En caso de que tenga menos de 18 años, mostrarle un mensaje que diga "No podés ingresar". Sea cual fuere la edad, el programa termina diciendo "Suerte".

### El código sería el siguiente:

```
1 Algoritmo seleccion_simple
2 Definir edad Como Entero;
3 Escribir "Ingresá tu edad:";
4 Leer edad;
5 Si (edad < 18) Entonces
6     Escribir "No podés ingresar"; //Se escribe según la edad
7 FinSi
8 Escribir "Suerte"; //Se escribe siempre
9 FinAlgoritmo
```

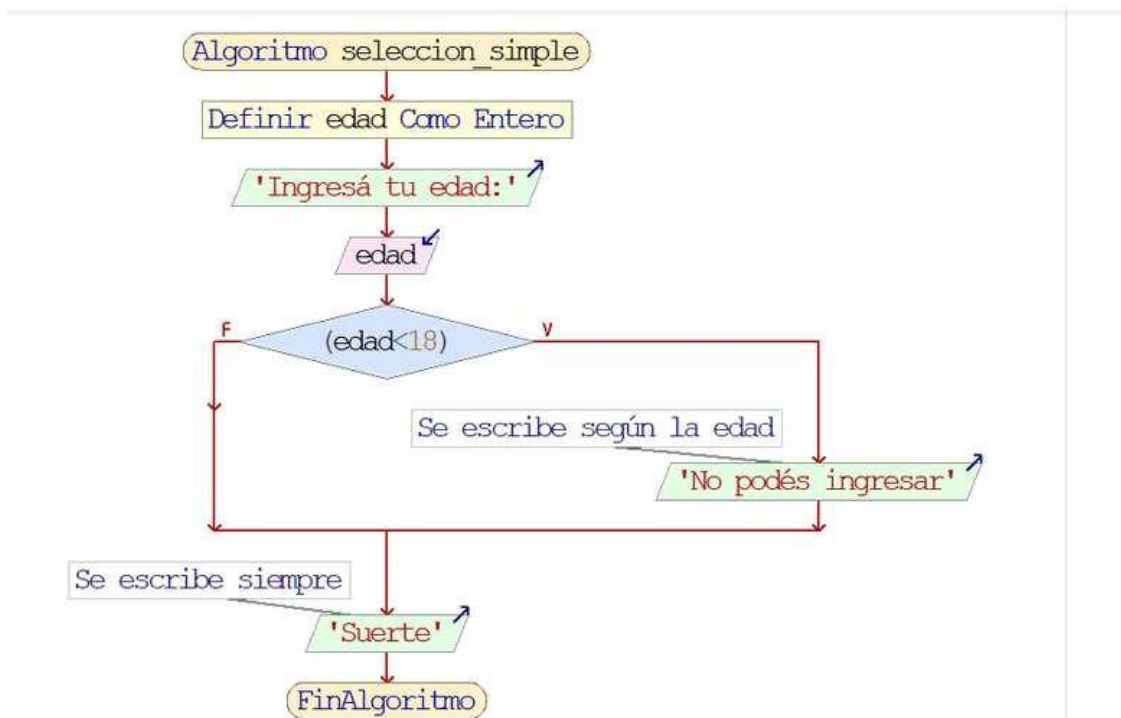
Si probamos el programa anterior con un valor igual o mayor a 18, por ejemplo, 30, la computadora evaluará la condición como FALSO, por lo que las instrucciones que están dentro del bloque Si...FinSi no serán ejecutadas. La ejecución continúa justo después del bloque Si...FinSi, encontrándose con la escritura de la palabra "Suerte" y dando por finalizada la ejecución.



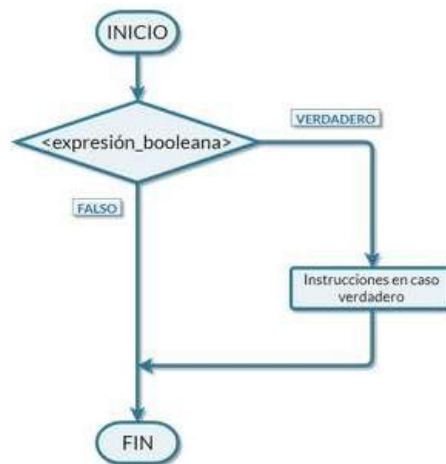
Si probamos el programa anterior con un valor menor a 18, por ejemplo, 12, la computadora evaluará la condición como VERDADERO, por lo que las instrucciones que están dentro del bloque Si...FinSi serán ejecutadas. La ejecución continúa justo después del bloque Si...FinSi, encontrándose con la escritura de la palabra "Suerte" y dando por finalizada la ejecución.

Es importante que notes que la palabra "Suerte" se escribe siempre, ya que al estar fuera del bloque Si...FinSi, su ejecución no se encuentra condicionada.

El diagrama de flujo que genera PSeInt es el siguiente:



### Estructura condicional simple



En cierto momento, definido por el programador, la computadora evalúa una condición, es decir, una expresión booleana, que se representa mediante un rombo. Si la expresión devuelve un resultado VERDADERO, la computadora ejecuta las instrucciones dentro de un bloque especial que luego retorna al flujo original, de lo contrario, el flujo continúa normalmente.

La sintaxis en pseudocódigo es la siguiente:

**Si (<expresión\_booleana>) Entonces**

**<instrucciones>**

**FinSi**

Si se tipea literalmente, indica que vamos a iniciar un bloque de selección.

<expresión\_booleana> es la condición que la computadora evaluará. El hecho de estar entre paréntesis es opcional, aunque te recomiendo que lo hagas para mejorar la legibilidad y porque muchos lenguajes formales requieren que la condición esté obligatoriamente encerrada entre paréntesis.

Entonces se tipea literalmente, indica que a continuación serán listadas las instrucciones que se ejecutarán en caso de que la condición sea VERDADERO.

<instrucciones> son las instrucciones que se ejecutarán. Puede ser solo una, o varias, siguiendo los conceptos vistos hasta aquí.

FinSi se tipea literalmente, indica que ha terminado el bloque de selección.

Un programa sencillo que facilita la comprensión podría ser el siguiente: pedirle al usuario que ingrese su edad. En caso de que tenga menos de 18 años, mostrarle un



mensaje que diga "No podés ingresar". Sea cual fuere la edad, el programa termina diciendo "Suerte".

**El código sería el siguiente:**

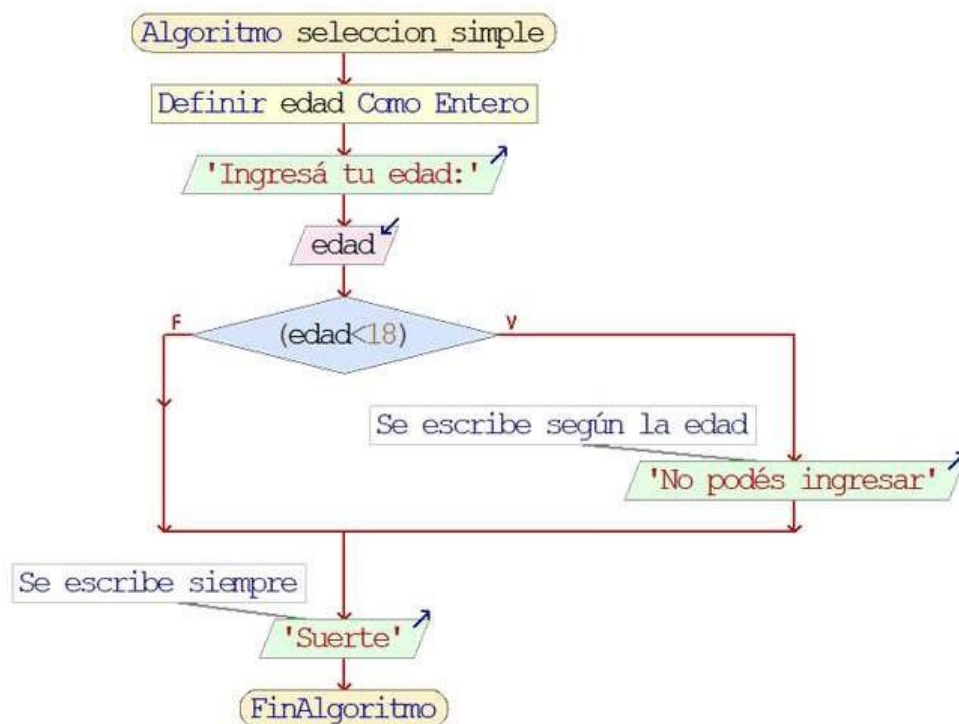
- 1 Algoritmo seleccion\_simple
- 2 Definir edad Como Entero;
- 3 Escribir "Ingresá tu edad:";
- 4 Leer edad;
- 5 Si (edad < 18) Entonces
- 6     Escribir "No podés ingresar"; //Se escribe según la edad
- 7 FinSi
- 8 Escribir "Suerte"; //Se escribe siempre
- 9 FinAlgoritmo

Si probamos el programa anterior con un valor igual o mayor a 18, por ejemplo, 30, la computadora evaluará la condición como FALSO, por lo que las instrucciones que están dentro del bloque Si...FinSi no serán ejecutadas. La ejecución continúa justo después del bloque Si...FinSi, encontrándose con la escritura de la palabra "Suerte" y dando por finalizada la ejecución.

Si probamos el programa anterior con un valor menor a 18, por ejemplo, 12, la computadora evaluará la condición como VERDADERO, por lo que las instrucciones que están dentro del bloque Si...FinSi serán ejecutadas. **La ejecución continúa justo después del bloque Si...FinSi, encontrándose con la escritura de la palabra "Suerte" y dando por finalizada la ejecución.**

Es importante que notes que la palabra "Suerte" se escribe siempre, ya que al estar fuera del bloque Si...FinSi, su ejecución no se encuentra condicionada.

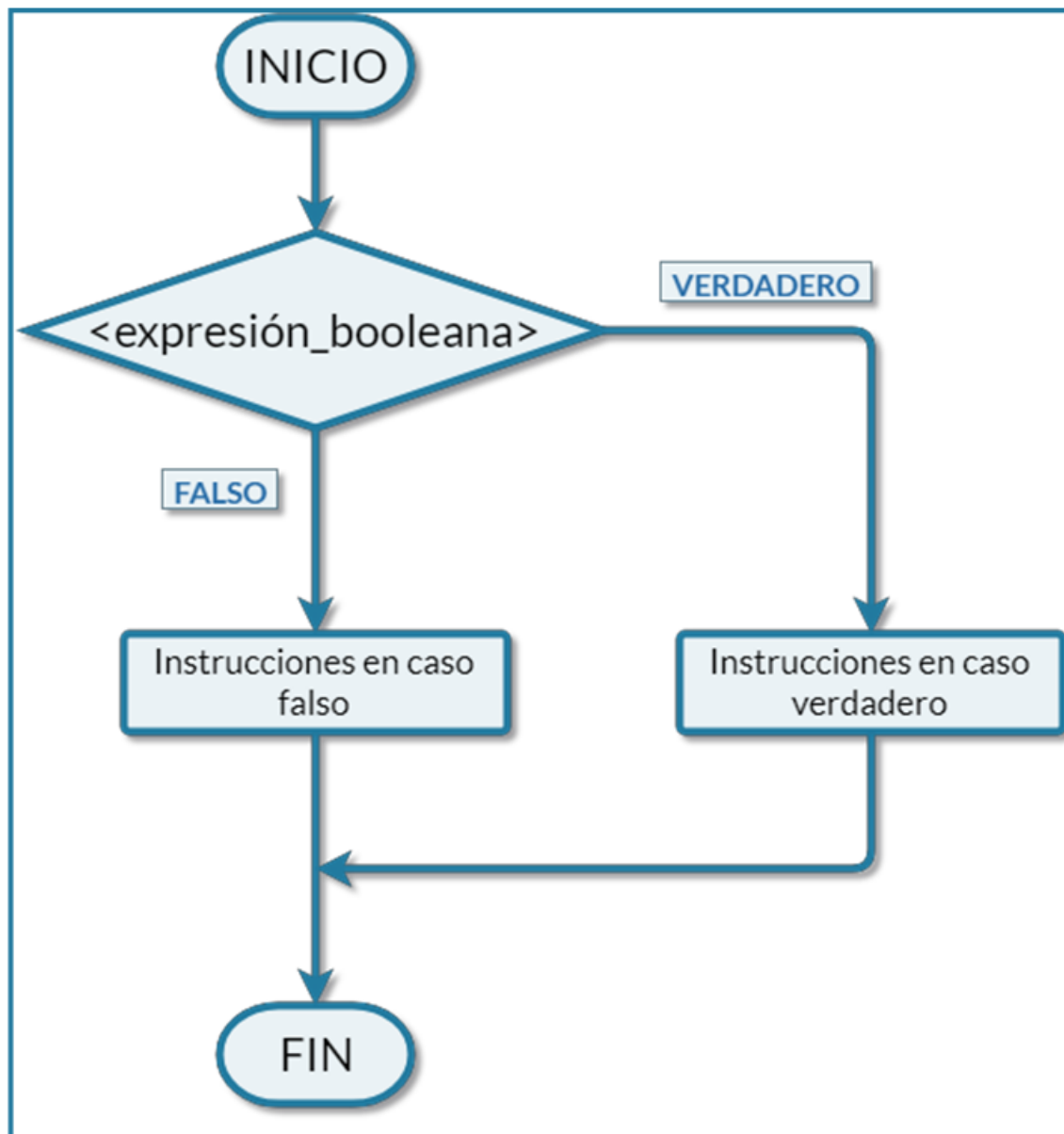
El diagrama de flujo que genera PSeInt es el siguiente:



Estructura condicional doble

La estructura anterior permite realizar una serie de instrucciones en caso de que una condición sea VERDADERO, pero no especifica nada en caso de que sea FALSO. De hecho, si la condición resulta FALSO, el programa continúa su ejecución como si el bloque Si...FinSi no existiera.

Para contemplar y hacer una serie de instrucciones específicas en caso de que una condición resulte FALSO, sin dejar de lado lo que se hacía cuando resultaba VERDADERO, es necesario utilizar una estructura de condicional doble, según el siguiente diagrama:



Si comparamos el diagrama de flujo de selección doble con el de selección simple. La única diferencia es que ahora sí podemos tomar acción en caso de que la condición resulte FALSO.

La sintaxis en pseudocódigo es la siguiente:





**Si (<expresión\_booleana>) Entonces**

**<instrucciones\_caso\_VERDADERO>**

**Sino**

**<instrucciones\_caso\_FALSO>**

**FinSi**

Si se tipea literalmente, indica que vamos a iniciar un bloque de selección.



**<expresión\_booleana>** es la condición que la computadora evaluará. El hecho de estar entre paréntesis es opcional, aunque recomendamos que lo hagan para mejorar la legibilidad y porque muchos lenguajes formales requieren que la condición esté obligatoriamente encerrada entre paréntesis.

Entonces se tipea literalmente, indica que a continuación serán listadas las instrucciones que se ejecutarán en caso de que la condición sea VERDADERO.

**<instrucciones\_caso\_VERDADERO>** son las instrucciones que se ejecutarán, si es que la condición resulta VERDADERO. Puede ser solo una, o varias, siguiendo los conceptos vistos hasta aquí.

Sino se tipea literalmente, indica que a continuación serán listadas las instrucciones que se ejecutarán en caso de que la condición sea FALSO.

**<instrucciones\_caso\_FALSO>** son las instrucciones que se ejecutarán, si es que la condición resulta FALSO. Puede ser solo una, o varias, siguiendo los conceptos vistos hasta aquí.

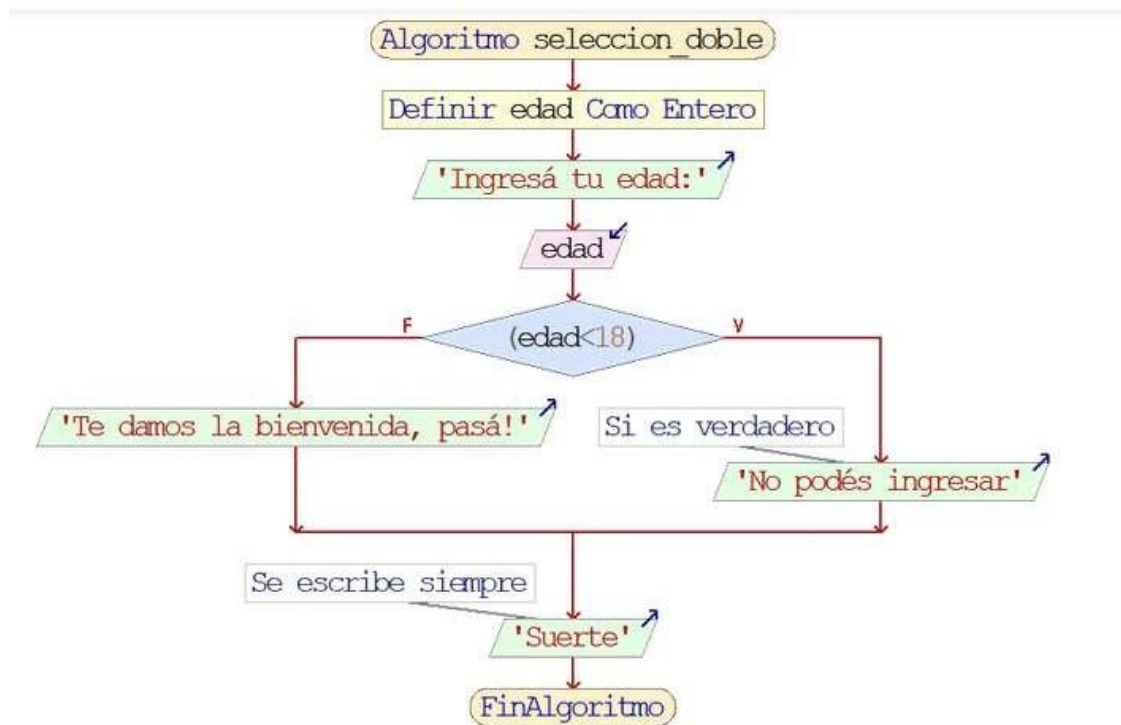
**FinSi** se tipea literalmente, indica que ha terminado el bloque de selección.

Continuando con el programa sencillo de ejemplo anterior, vamos a pedirle al usuario que ingrese su edad. En caso de que tenga menos de 18 años, mostrarle un mensaje que diga "No podés ingresar", en caso contrario, dirá "Te damos la bienvenida, ¡pasá!". Sea cual fuere la edad, el programa termina diciendo "Suerte".

El código sería el siguiente

- 1 Algoritmo de seleccion\_doble
- 2 Definir edad Como Entero;
- 3 Escribir "Ingresá tu edad:";
- 4 Leer edad;
- 5 Si (edad < 18) Entonces
- 6     Escribir "No podés ingresar"; //Si es VERDADERO
- 7 Sino
- 8     Escribir "Te damos la bienvenida, ¡pasá!";
- 9 FinSi
- 10 Escribir "Suerte"; //Se escribe siempre
- 11 FinAlgoritmo

El diagrama de flujo que genera PSeInt es el siguiente:



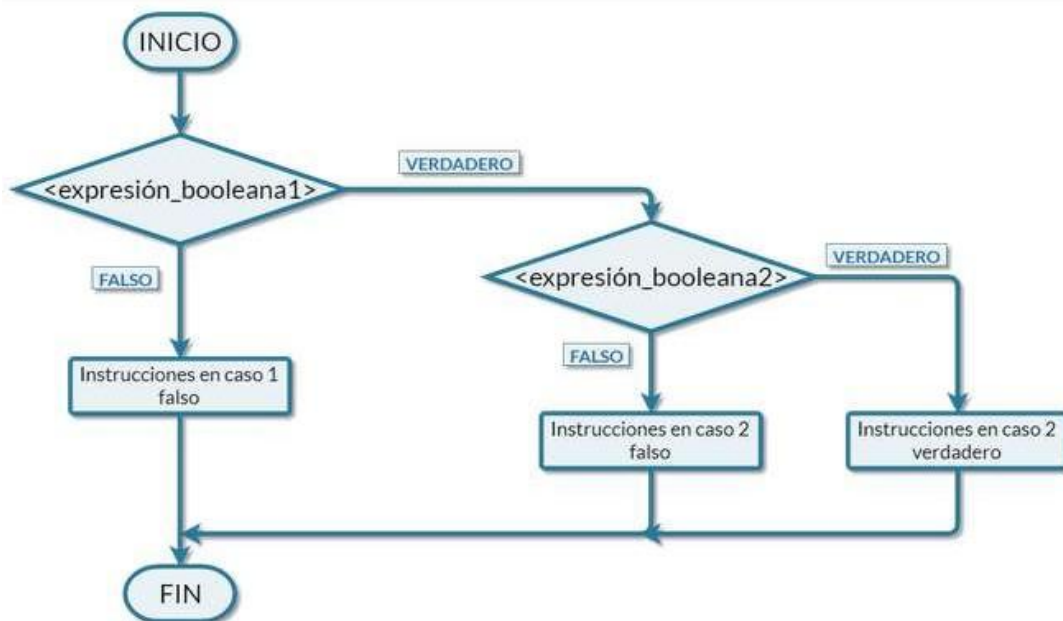
Si probamos el programa anterior con un valor igual o mayor a 18, por ejemplo, 30, la computadora evaluará la condición como FALSO, por lo que se ejecutarán las instrucciones que estén justo después del Sino. La ejecución continúa justo después del bloque Si...FinSi, encontrándose con la escritura de la palabra "Suerte" y dando por finalizada la ejecución.

Si probamos el programa anterior con un valor menor a 18, por ejemplo, 12, la computadora evaluará la condición como VERDADERO, por lo que se ejecutarán las instrucciones que estén justo después del Si, concluyendo justo antes del Sino. La ejecución continúa justo después del bloque Si...FinSi, encontrándose con la escritura de la palabra "Suerte" y dando por finalizada la ejecución.

Es importante notar que la palabra "Suerte" se escribe siempre, ya que al estar fuera del bloque Si...FinSi, su ejecución no se encuentra condicionada.

Para poder realizar programas más complejos, que contemplen más posibilidades (o caminos), basta con poder anidar estructuras de selección, cuestión que veremos a continuación.

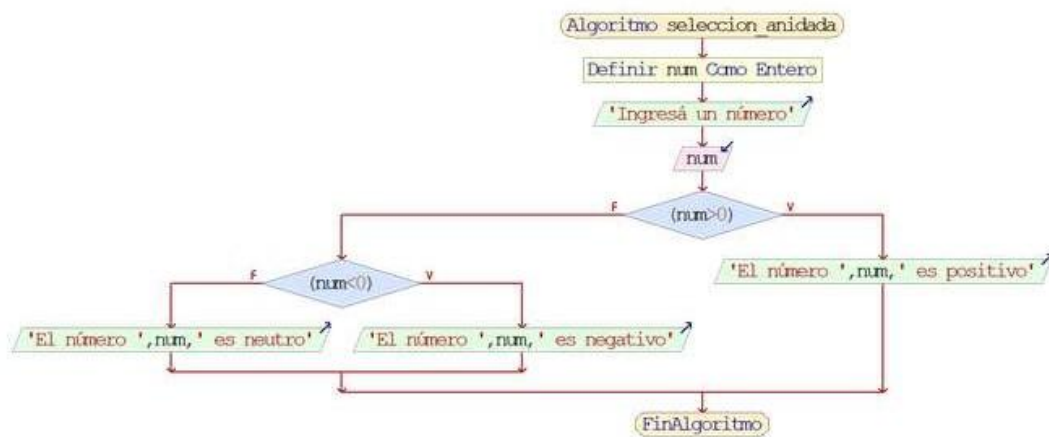
## Anidamiento de estructuras de condición



Para comprender este tema, la idea es realizar el siguiente algoritmo: el usuario ingresa un número y la computadora indica si se trata de un número positivo, negativo o el cero.

Habrán visto que la máquina solo puede resolver expresiones booleanas, las cuales solo tienen dos resultados posibles: VERDADERO o FALSO. Como nuestro programa evidentemente contempla tres posibles acciones, debemos hacer uso de anidamiento de estructuras de selección, obteniendo un flujo como el siguiente:

Un ejemplo puede ser el siguiente: si el número ingresado es mayor que 0, se trata sin discusión de un número positivo, por lo tanto, suponiendo que alojo el número en una variable llamada num, la expresión  $\text{num} > 0$  resultaría VERDADERO. En caso FALSO, no se puede asegurar que se trate de un número negativo, pues pudo haber sido el 0. En ese caso, se tiene que volver a emplear una estructura de selección, cuya condición puede ser  $\text{num} < 0$ . Si la expresión anterior resulta VERDADERO, se puede asegurar que se trata de un número negativo, en caso contrario, puedo asegurar que se trata del 0.



### Diagrama de código:

- 1 Algoritmo seleccion\_anidada
- 2 Definir num Como Entero;
- 3 Escribir "Ingresá un número";
- 4 Leer num;
- 5 Si (num > 0) Entonces
- 6 Escribir "El número",num,"es positivo";
- 7 SiNo
- 8 Si (num < 0) Entonces
- 9 Escribir "El número",num,"es negativo";
- 10 SiNo
- 11 Escribir "El número",num,"es neutro";
- 12 FinSi
- 13 FinSi
- 14 FinAlgoritmo



Por supuesto que el ejemplo anterior puede hacerse de muchas formas. Por ejemplo, podemos primero poner una condición `num == 0`. Si el resultado es VERDADERO, el número es el 0, de lo contrario, debo preguntar por alguna de las dos posibilidades: si es mayor, o si es menor. Eligiendo la segunda. Si resulta VERDADERO, el número es negativo, de lo contrario, es positivo.

### Estructuras de control iterativas

En todos los ejemplos que vimos previamente las instrucciones se realizan una única vez.

**Te pedimos que vuelvas al algoritmo que vimos en la sección Condicional alternativo, donde se informa si la persona es o no mayor de edad.**

Este algoritmo pide una sola edad, es decir que, cada vez que se ejecute, funcionará para una sola persona. Seguramente estarás pensando que sería mucho mejor que se pudieran ingresar las edades de muchas personas, sin tener la necesidad de ejecutarlo nuevamente.

En otras palabras, sería útil que hiciera lo mismo que hace para una persona, pero para muchas, de forma reiterativa.

Por lo tanto, se presenta la necesidad de tener que ejecutar una o más acciones más de una vez, de forma reiterativa. En este caso haremos uso de las estructuras de control iterativas.

Nuestro lenguaje pseudocódigo nos permite controlar las repeticiones de distintas maneras y es por eso que a continuación veremos **tres estructuras de control iterativas**.

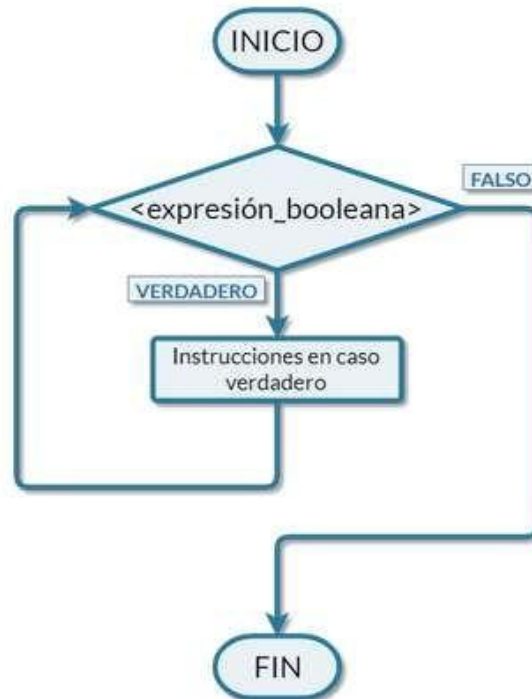
#### 1. Ciclo Mientras

Una estructura de repetición o iterativa nos permite especificar que un programa debe repetir una o más acciones mientras cierta condición valga VERDADERO. Cuando la condición pase a valer FALSO, el programa continuará con la ejecución de las demás sentencias debajo de la estructura de control de repetición.

Supongamos que queremos escribirle al usuario la frase "Hola, mucho gusto." unas cinco veces. Es cierto que una posibilidad es usar cinco instrucciones Escribir, una debajo de otra con un resultado satisfactorio. Pero es evidente que esto no es muy práctico. Peor

aún, imaginemos que olvidamos poner un punto al final de la oración. ¿Se imaginan cambiándolo en las cinco instrucciones? ¿Y si hubiésemos querido mostrar diez mil oraciones? Aquí es cuando una estructura de repetición nos salva.

Observemos el siguiente diagrama:



La sintaxis en pseudocódigo es la siguiente:

**Mientras (<expresión\_booleana>) Hacer**

**<instrucciones>**

**FinMientras**

**Mientras** se tipea literalmente, indica que vamos a iniciar un bloque de repetición.

**<expresión\_booleana>** es la condición que la computadora evaluará. El hecho de estar entre paréntesis es opcional, aunque se recomienda hacerlo para mejorar la legibilidad y porque muchos lenguajes formales requieren que la condición esté obligatoriamente encerrada entre paréntesis.

**Hacer** se tipea literalmente, indica que a continuación serán listadas las instrucciones que se ejecutarán en caso de que la condición sea VERDADERO.



**<instrucciones>** son las instrucciones que se ejecutarán. Puede ser solo una, o varias, siguiendo los conceptos vistos hasta aquí.

**FinMientras** se tipea literalmente, indica que ha terminado el bloque de repetición.

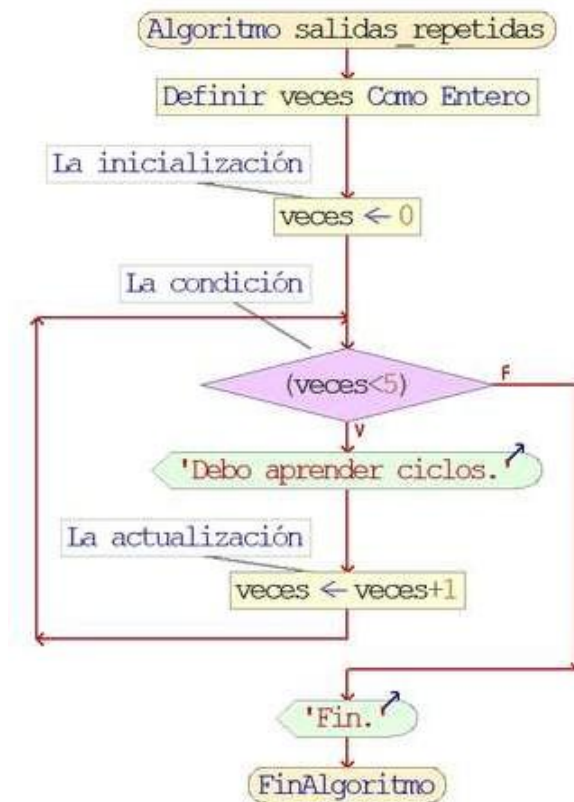
Tomando como ejemplo que se le quiere mostrar al usuario la frase "Debo aprender ciclos." cinco veces, el código puede ser el siguiente:

**Código:**

- 1 Algoritmo salidas\_repetidas
- 2 Definir veces Como Entero;
- 3 veces = 0; //La inicialización
- 4 Mientras (veces < 5) Hacer //La condición
- 5 Escribir "Debo aprender ciclos.";
- 6 veces = veces + 1; //La actualización
- 7 FinMientras
- 8 Escribir "Fin.";
- 9 FinAlgoritmo

Resultando en el siguiente diagrama de flujo:





Lo primero que hicimos fue definir una variable entera llamada veces, cuyo valor inicial es cero.

Es muy importante que veces tenga un valor inicial, pues a continuación, su valor es comparado con el número 5. Si el valor de veces es menor que 5, se ejecutan las instrucciones dentro del bloque Mientras...FinMientras. Dentro del bloque, se escribe la cadena "Debo aprender ciclos.". Acto seguido, se incrementa una unidad el valor de veces, a través de su valor actual.

Una vez que se llega a la sentencia FinMientras, el flujo vuelve hacia arriba a evaluar la condición. El valor de veces ya no es 0, porque fue actualizado antes de terminar la primera iteración. Ahora su valor es 1. Como el valor 1 es menor que 5, se vuelven a realizar las instrucciones: mostrar la cadena e incrementar el valor de veces. Esto se repite hasta el caso donde veces valga 5. Cuando la computadora evalúe la condición  $(veces < 5)$ , el resultado será FALSO. En tal situación, el bloque de repetición termina, continuando con el normal flujo secuencial.

La ausencia o el mal cálculo de alguno de los tres componentes que protagonizan un ciclo (inicialización, condición y actualización) pueden provocar resultados inesperados: quizás se realicen más o menos iteraciones de las que fueron previstas, que el ciclo nunca se ejecute (la primera iteración resulta **FALSO**) o que el ciclo se ejecute infinitamente (todas las iteraciones resultan **VERDADERO**)

En la inicialización, podemos probar estableciendo el valor de veces en 1, para que se observe que solo se imprime cuatro veces la cadena "Debo aprender ciclos."

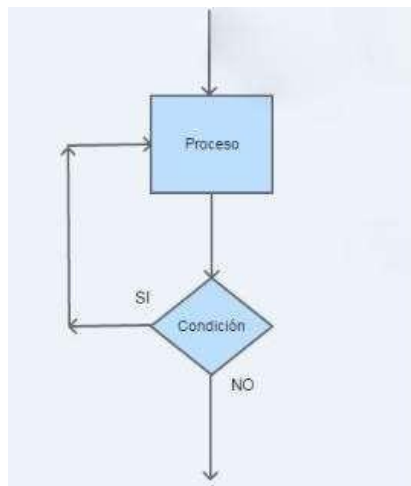
En la condición, podemos probar cambiando el operador < por un >, para que se vea que el ciclo nunca se ejecutará, pues la primera vez, veces vale 0 y tal valor no es mayor que 5, haciendo que la condición resulte FALSO y continuando por debajo de la estructura de repetición.

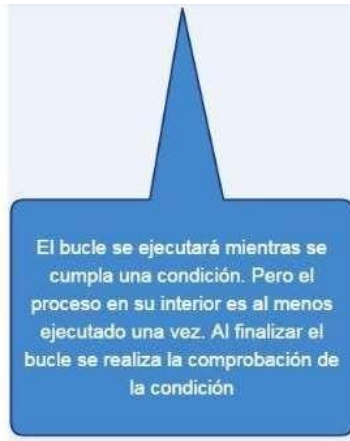
En la actualización se puede probar con borrar o poner en comentario la instrucción `veces = veces + 1`, para que veamos que el ciclo se ejecuta infinitamente, debido a que veces vale constantemente 0, haciendo que la condición resulte VERDADERO en todas las iteraciones.

Queda claro que el uso de estructuras de repetición requiere de una atención especial. Es muy fácil que nuestro programa no tenga errores de sintaxis, pero sí lógicos, es decir, que haya cosas que no funcionen como esperamos o que den resultados inesperados. Como todo, es cuestión de tomarle la mano. Gracias a los ciclos, mostrar una frase cinco, diez, cien o un millón de veces es tan simple como cambiar el número con el cual se compara veces en la condición.

## 2. Ciclo Repetir-Hasta que

Veamos ahora qué sucede con este otro ciclo Repetir y para eso te proponemos reflexionar sobre el siguiente caso:





## SEUDOCÓDIGO

INICIO

Repetir

Mostrar ("Ingrese un número mayor a 0")

Ingresar (NUM)

Hasta que (NUM > 0)

Mostrar ("Ingresó el número:", NUM)

FIN

Es muy similar al caso anterior, pero cuidado, porque los pequeños cambios son realmente muy significativos y originan la diferencia de comportamiento entre el "Mientras" y el "Repetir".

Lo primero que encontramos es la acción **Repetir**, lo que produce que "entre" al bucle y se ejecuten las acciones que se encuentran hasta el momento de encontrar la frase "Hasta que". Luego se encuentra una condición que habrá que evaluar.

En el caso de que la condición sea **FALSA**, se realiza una nueva iteración del ciclo, es decir, que vuelve a repetir las acciones que se encuentran entre las palabras "**Repetir**" y "**Hasta Que**". Si en cambio, la condición es **VERDADERA**, se da por finalizado el ciclo y continúa con la próxima acción.

La estructura genérica de un **Repetir** es la siguiente:

## SEUDOCÓDIGO

Repetir

Instrucción1

..... Condición FALSA

InstrucciónN

Hasta que (CONDICIÓN)

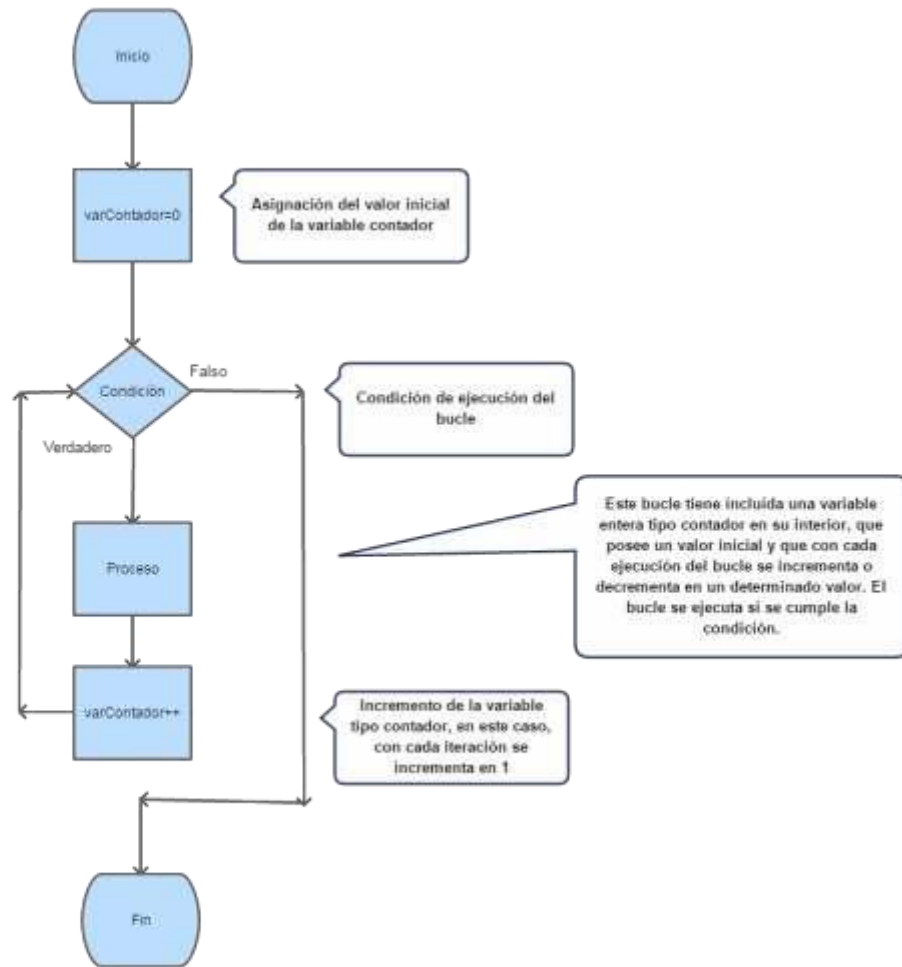


Antes de continuar, repasemos las características del **Repetir**:

- Se evalúa la condición luego de ejecutar el bloque.
- Cuando la condición es Falsa se ejecutan las acciones del cuerpo del bucle.
- Cuando la condición es Verdadera finaliza el bucle.
- El cuerpo del bucle se ejecuta al menos una sola vez, ya que, si bien inicialmente la condición puede ser Verdadera, se evalúa después de ejecutarse el bloque.

## Ciclo Para

Veamos ahora otro ciclo, pero con características distintas a los ciclos que te presentamos anteriormente.



Antes de continuar con la explicación, te proponemos que analices este ejemplo para que puedas deducir qué es lo que hace.



#### SEUDOCÓDIGO

##### VARIABLES

N : ENTERO

##### INICIO

Para N <- 1 hasta 10 Hacer

Mostrar (N)

FinPara

##### FIN

¿Qué tal resultó el análisis?

*Hacé clic en el botón para ver la respuesta.*

Seguramente estarás pensando que se muestran los números del 1 al 10. Si es así, ¡felicitaciones! Porque este ciclo realiza precisamente eso.




También es importante que comprendas que los valores iniciales y finales son también válidos, es decir que la primera iteración se ejecutará con  $N=1$ , y la última iteración se ejecutará con  $N=10$ .

### ¿Resolvemos el algoritmo de notas y promedios con un Para?

Otra característica de este **ciclo Para** es que nosotros sabemos cuántas veces se va a ejecutar el ciclo, ya que debemos especificar el valor inicial y final que tendrá la variable del ciclo.

En el ejemplo que vimos en la sección anterior,


 ¿Cuántas veces se ejecuta el Para?

Se ejecuta diez veces, ya que N toma los valores desde 1 hasta 10.

La estructura genérica de un **Para** es la siguiente:

#### SEUDOCÓDIGO

```
Para N <- ValorInicial hasta ValorFinal Hacer
    Instrucción1
    .....
    InstrucciónN
FinPara
```

 Destacamos algunas características del **Para**:

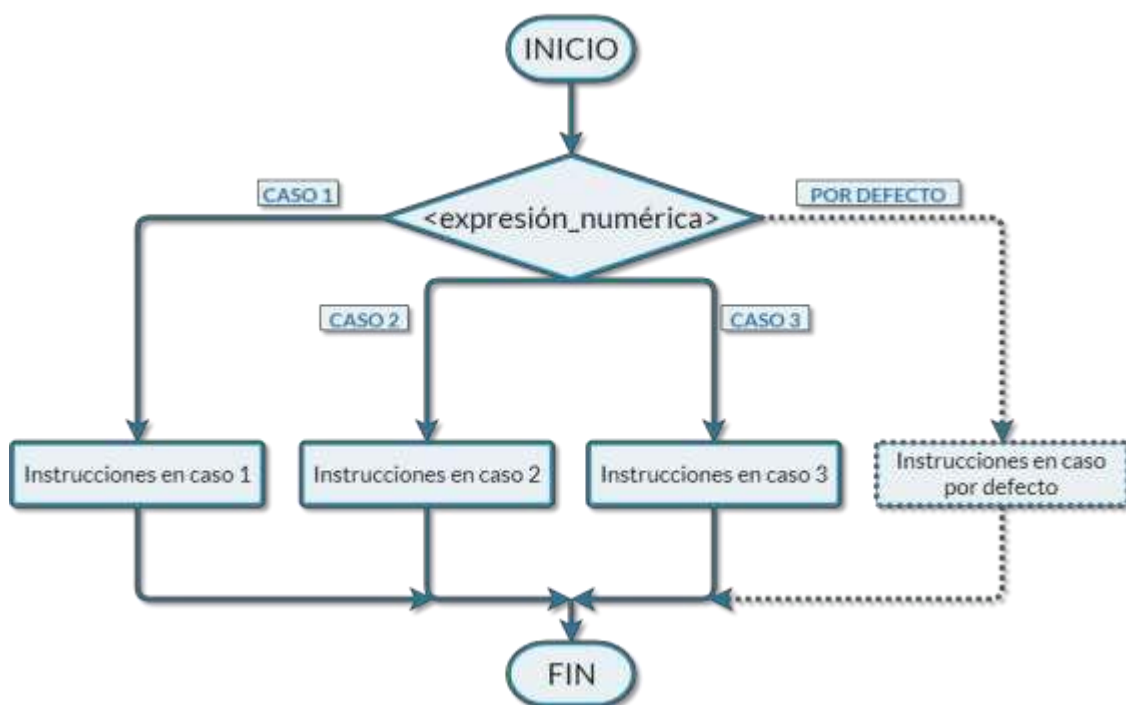
- La variable del índice del Para debe ser una Variable de tipo Entero.
- **ValorInicial** y **ValorFinal** deben ser variables o expresiones de tipo entero.
- ValorInicial debe ser menor o igual a ValorFinal para que el ciclo se ejecute, de lo contrario no se ejecutará.
- La cantidad de veces que se ejecuta un ciclo "Para" es  $\text{ValorFinal} - \text{ValorInicial} + 1$ .
- Dentro del cuerpo del ciclo es un error grave modificar el valor del índice o bien del ValorInicial o del ValorFinal.

Ahora, hagamos una modificación al algoritmo ya conocido de las notas. El cambio será que, al comenzar el programa, le preguntaremos al usuario cuántos alumnos quiere ingresar. Y de ese modo realizaremos el ciclo la cantidad de veces que nos diga el usuario, ¿te animás a proponer el algoritmo?



```
1  Algoritmo CalcularPromedio
2      definir NOTA1, NOTA2, NOTA3, I, CANT_ALUMNOS como entero
3      definir PROMEDIO como real
4      definir NOMBRE como CADENA
5
6      Escribir "Ingrese la cantidad de alumnos"
7      Leer CANT_ALUMNOS
8
9
10     Para I<-1 Hasta CANT_ALUMNOS Con Paso 1 Hacer
11         Escribir "Ingrese el nombre del alumno"
12         Leer NOMBRE
13         Escribir "Ingrese nota 1"
14         Leer NOTA1
15         Escribir "Ingrese nota 2"
16         Leer NOTA2
17         Escribir "Ingrese nota 3"
18         Leer NOTA3
19
20         PROMEDIO<-(NOTA1+NOTA2+NOTA3)/3
21         Escribir "El promedio de notas es:",PROMEDIO
22         Si PROMEDIO >= 4 entonces
23             Escribir NOMBRE," Aprobó la materia."
24         Sino
25             Escribir NOMBRE," Desaprobó la materia."
26         FinSi
27     FinPara
28 FinAlgoritmo
```

## Estructura de control selectiva



Supongamos que deseamos realizar un menú de opciones, como cuando llamamos por teléfono y la voz del otro lado dice: "Para ventas, presione 1. Para pagos, presione 2. Para servicio técnico, presione 3...". Realizar un menú con, por ejemplo, diez opciones diferentes, requiere, como hemos visto, realizar un anidamiento de estructuras de selección que puede tornarse inmanejable, engorroso y poco legible.





Para los casos en los que el dato a evaluar sea un número, y además por comparación (es decir, cada opción tiene un número concreto, no un rango de valores) se puede hacer uso de una estructura de selección múltiple, la cual presenta un diagrama como el siguiente:

La sintaxis es la siguiente:

**Segun (<variable\_numérica>) Hacer**

**<opcion>:**

**<instrucciones>**

**<opcion>:**

**<instrucciones>**

**<opcion>:**

**<instrucciones>**

**...**

**De Otro Modo:**

**<instrucciones>**

**FinSegun**

**Segun** se tipea literalmente, indica que vamos a iniciar un bloque de selección múltiple.

**<variable\_numérica>** es la variable que la computadora evaluará. El hecho de estar entre paréntesis es opcional, aunque recomendamos que lo hagan para mejorar la legibilidad y porque muchos lenguajes formales requieren que la condición esté obligatoriamente encerrada entre paréntesis.

**Hacer** se tipea literalmente, indica que a continuación serán listadas las posibilidades que puede tomar la **<variable\_numérica>**.

**<opcion>** es un valor numérico que se estima puede llegar a valer la

**<variable\_numérica>**. El **:** luego de cada **<opcion>** es obligatorio e indica que a continuación se listarán las instrucciones para ese caso.

**<instrucciones>** son las instrucciones que se ejecutarán si la opción asociada se cumple.



Los ... indican que pueden seguir listándose posibles opciones, cada una con su correspondiente lista de instrucciones a ejecutarse.

**De Otro Modo:** se tipea literalmente, aunque es opcional. Se listan las instrucciones que se ejecutarán en caso de que ninguna de las opciones contempladas ocurra.

**FinSegun** se tipea literalmente, indica que ha terminado el bloque de selección múltiple.

Hagamos el menú, con las siguientes opciones:

Ventas.

Pagos.

Servicio técnico.

Gerencia.

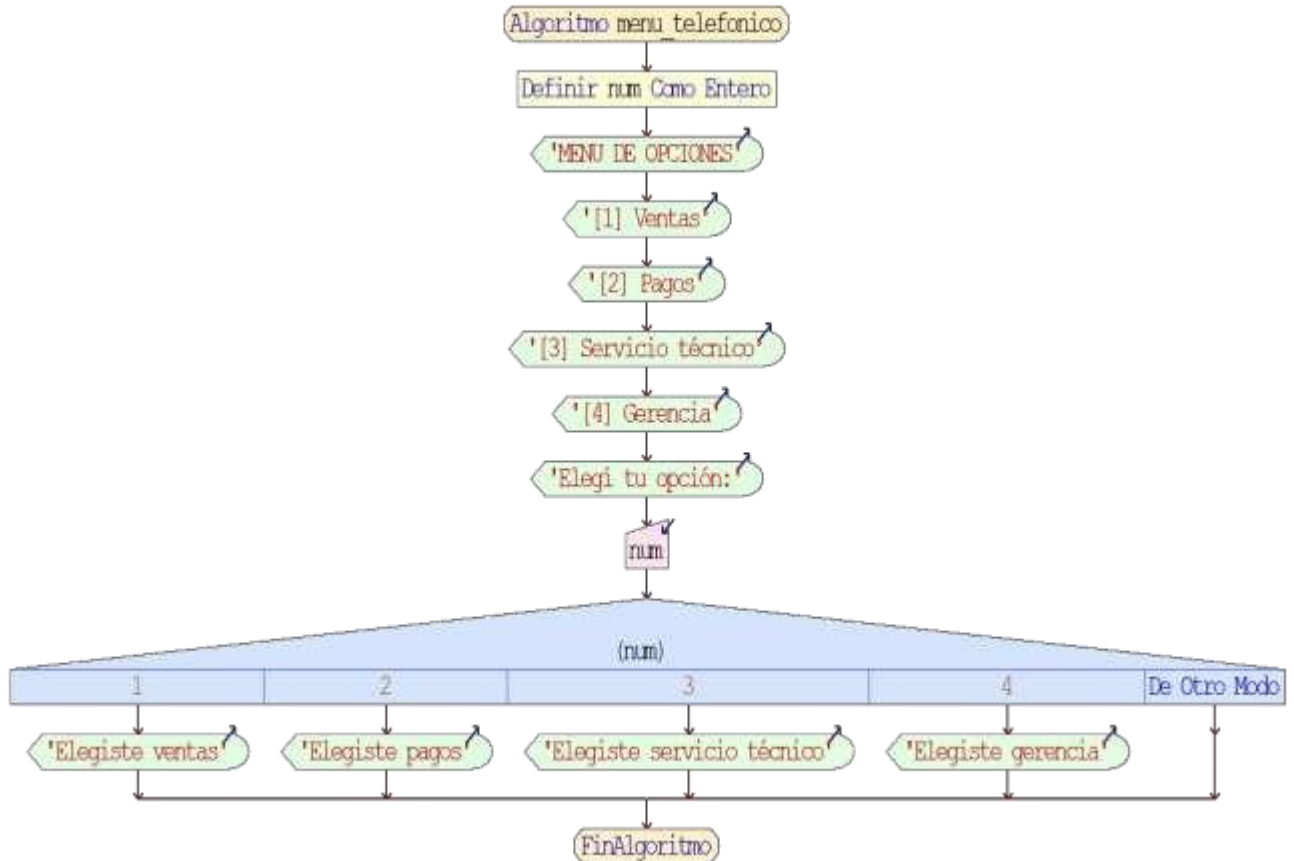
Lo más conveniente es hacer uso de Segun...FinSegun:

#### Menú de Opciones

```
1  Algoritmo menu_telefonico
2  Definir num Como Entero;
3  Escribir "MENU DE OPCIONES";
4  Escribir "[1] Ventas";
5  Escribir "[2] Pagos";
6  Escribir "[3] Servicio técnico";
7  Escribir "[4] Gerencia";
8  Escribir "Elegí tu opción:";
9  Leer num;
10 Segun (num)
11      Hacer 1:
12          Escribir "Elegiste ventas";
13      2:
14          Escribir "Elegiste pagos";
15      3:
16          Escribir "Elegiste servicio técnico";
17      4:
18          Escribir "Elegiste gerencia";
19 FinSegun
```

## 20 FinAlgoritmo

Diagrama de flujo:



Si el usuario ingresa un valor no contemplado entre las opciones dentro del bloque

Segun...FinSegun, el programa simplemente continúa su ejecución.

Poner una variable no numérica (como una cadena o un lógico) para ser evaluada por un bloque **Segun...FinSegun** derivará en un error en tiempo de ejecución.

En este caso, agregaremos a este ejemplo la posibilidad de que la máquina realice instrucciones en caso de que el número ingresado no sea ninguno de los listados. Para ello, haremos uso de la sentencia De Otro Modo. La computadora responderá "Incorrecto" si se llega a tal caso.

Menú de opciones mejorado

- 1 Algoritmo menu\_telefonico\_mejorado
- 2 Definir num Como Entero;
- 3 Escribir "MENU DE OPCIONES";
- 4 Escribir "[1] Ventas";
- 5 Escribir "[2] Pagos";
- 6 Escribir "[3] Servicio técnico";



```
7  Escribir "[4] Gerencia";
8  Escribir "Elegí tu opción:"
9  Leer num;
10 Segun (num)
11      Hacer 1:
12          Escribir "Elegiste ventas";
13      2:
14          Escribir "Elegiste pagos";
15      3:
16          Escribir "Elegiste servicio técnico";
17      4:
18          Escribir "Elegiste gerencia";
19      De Otro Modo:
20          Escribir "Incorrecto";
21 FinSegun
22 FinAlgoritmo
```

El uso de la sentencia De Otro Modo es opcional.

## Variables de tipo contador y variables tipo acumulador

Tomando como ejemplo el siguiente caso: **“Calcular el promedio de una materia para un alumno. Dicha materia compone su calificación final de las notas de tres exámenes”**

Para calcular el promedio tenemos que sumar las tres notas y dividir las por 3, ya que corresponde a la cantidad de notas del alumno. De forma genérica, *un promedio se obtiene dividiendo el total acumulado por la cantidad de elementos*. De esta manera, surge la necesidad de contar la cantidad de notas y, por otro lado, la de acumular la suma de esas notas.

Hagamos una nueva versión del algoritmo original:

```
VARIABLES
    NOTA1, NOTA2, NOTA3, TOTAL_NOTAS, CANT_NOTAS : ENTERO
    PROMEDIO : REAL
INICIO
    TOTAL_NOTAS <- 0
    CANT_NOTAS <- 0
    Mostrar ("Ingreso nota 1")
    Ingresar (NOTA1)
    TOTAL_NOTAS <- TOTAL_NOTAS + NOTA1
    CANT_NOTAS <- CANT_NOTAS + 1
    Mostrar ("Ingreso nota 2")
    Ingresar (NOTA2)
    TOTAL_NOTAS <- TOTAL_NOTAS + NOTA2
    CANT_NOTAS <- CANT_NOTAS + 1
    Mostrar ("Ingreso nota 3")
```



```
Ingresar (NOTA3)
TOTAL_NOTAS <- TOTAL_NOTAS + NOTA3
CANT_NOTAS <- CANT_NOTAS + 1
PROMEDIO <- TOTAL_NOTAS / CANT_NOTAS
Mostrar ("El promedio de las notas es:", PROMEDIO)
FIN
```

Como habrás observado, tenemos dos variables: **TOTAL\_NOTAS** y **CANT\_NOTAS**.

En **TOTAL\_NOTAS** iremos **acumulando la sumatoria de todas las notas**. Para acumular las notas comenzamos en 0 y luego, por cada nota que se ingresa, sumamos ese valor al total acumulado.

Del mismo modo iremos contando la cantidad de notas en la variable **CANT\_NOTAS**. Para **contar las notas** comenzamos en 0 y luego, por cada nota que se ingresa, contamos esa nota sumándole 1 a **CANT\_NOTAS**.

El comportamiento que toman las variables que van acumulando y contando se conoce en programación como:

**TOTAL\_NOTAS:** Acumulador  
**CANT\_NOTAS:** Contador.

Los acumuladores y contadores son muy comunes en programación y seguramente los vas a utilizar en muchos de tus algoritmos.

**Veamos otro ejemplo:**

ENTRADAS\_VENDIDAS = 30

ENTRADAS\_VENDIDAS = ENTRADAS\_VENDIDAS + 1

Veamos qué sucede con estas instrucciones...

ENTRADAS\_VENDIDAS toma el valor 30

¿Qué valor tendrá ENTRADAS\_VENDIDAS luego de la asignación?

ENTRADAS\_VENDIDAS tendrá el valor 30 + 1, es decir, 31.

Esta **asignación** que acabamos de realizar (sumarle o restarle un valor constante a una variable), es lo que en programación se conoce como **Contado**

**Veamos un ejemplo más:**

MONTO\_RECAUDADO = 150

PRIMERA\_VENTA = 35

MONTO\_RECAUDADO = MONTO\_RECAUDADO + PRIMERA\_VENTA

SEGUNDA\_VENTA = 40



MONTO\_RECAUDADO = MONTO\_RECAUDADO +  
SEGUNDA\_VENTA

Veamos qué sucede con estas instrucciones...  
MONTO\_RECAUDADO toma el valor 150.  
PRIMERA\_VENTA toma el valor 35.  
MONTO\_RECAUDADO toma el valor de MONTO\_RECAUDADO + el valor de  
PRIMERA\_VENTA.

Es decir  $150 + 35$ . Por lo tanto, MONTO\_RECAUDADO tendrá el valor 185

SEGUNDA\_VENTA toma el valor 40.  
MONTO\_RECAUDADO toma el valor de MONTO\_RECAUDADO + el valor de  
SEGUNDA\_VENTA.

Es decir  $185 + 40$ . Por lo tanto, MONTO\_RECAUDADO tendrá el valor 225

En este caso, a una variable se le está acumulando su mismo valor con el valor de otra variable.

Esta operación es lo que en programación se conoce como **Acumulado**.

### Operadores lógicos

Son aquellos que permiten operar con expresiones booleanas. A través de ellos, pueden realizarse condiciones más complejas que no requieran un anidamiento de estructuras de selección.

Así como los operadores aritméticos tienen como operandos a números, los operadores lógicos tienen como operandos a valores lógicos (VERDADERO o FALSO).

Así como los operadores aritméticos devuelven como resultado un número, los operadores lógicos devuelven como resultado un valor lógico (VERDADERO o FALSO).

Operador	Nombre
!	NO lógico (NOT)
&	Y lógico (AND)
	O lógico (OR)

Cada uno de estos operadores tiene su tabla de verdad, es decir, una representación de todas las combinaciones posibles con sus correspondientes resultados. A continuación, se detallan cada uno de estos operadores.



## Operador NOT

El operador NOT, también llamado NO o negación lógica, a diferencia de los que veníamos observando, es un operador monádico. Esto significa que trabaja con un solo operando. Lo que hace este operador es negar la expresión a la cual afecta. Se representa con el símbolo !.

Probemos el siguiente código:

```
1                               Algoritmo                               not
2                               Escribir                               2      <      3;
3                               Escribir                               !(2    <      3);
4  FinAlgoritmo
```

La expresión  $2 < 3$  es verdadera, por lo tanto, el primer Escribir muestra VERDADERO. En cambio, la expresión lógica en el segundo Escribir está siendo negada, por lo tanto, se devuelve lo contrario (FALSO).

Los paréntesis en la segunda salida no son obligatorios, pero ayudan a la legibilidad. La tabla de verdad del operador NOT es la siguiente:

Expresión booleana	Resultado
!VERDADERO	FALSO
!FALSO	VERDADERO

## Operador OR

El operador OR, también llamado O o disyunción lógica, trabaja con dos operandos. Lo que hace este operador es devolver un valor VERDADERO si al menos un operando es VERDADERO, en caso contrario devuelve FALSO. Se representa con el símbolo |.

Ejemplo: La condición para pasar a cierto evento es ser mayor de edad o tener un nombre cuya longitud no supere los seis caracteres. Haré un programa que permita al usuario ingresar su nombre y su edad. Luego la máquina informará si puede pasar o no, según la condición planteada.

El código puede ser el siguiente:

```
1  Algoritmo or
2  Definir nombre ComoCadena;
3  Definir edad ComoEntero;
4  Escribir "Ingresá tu nombre";
```



- 5 Leer nombre;
- 6 Escribir "Ingresá tu edad";
- 7 Leer edad;
- 8 Si (Longitud(nombre) <= 6 | edad >= 18) Entonces
- 9 Escribir "Puede pasar";
- 10 SiNo
- 11 Escribir "NO Puede pasar";
- 12 FinSi
- 13 FinAlgoritmo

Observemos las siguientes pruebas:

Nombre	Edad	Salida
"Carlos"	25	"Puede pasar"
"María"	14	"Puede pasar"
"Ricardo"	40	"Puede pasar"
"Daniela"	8	"NO Puede pasar"

Para formalizar, la tabla de verdad del operador OR es la siguiente:

Expresión booleana	Resultado
VERDADERO   VERDADERO	VERDADERO
VERDADERO   FALSO	VERDADERO
FALSO   VERDADERO	VERDADERO

Una buena aplicación práctica de este operador podría ser para pedirle una respuesta al usuario. Supongamos una aplicación que requiera que el usuario ingrese la palabra "Si" para continuar. Es evidente que puede ingresarlo todo en minúsculas, todo en mayúsculas o de forma alternada. Para contemplar todas estas combinaciones y que la máquina las interprete como la misma opción, podemos armar una condición con operadores lógicos.

Observemos el siguiente código:

- 1 Algoritmo or2
- 2 Definir opcion Como Cadena;
- 3 Escribir "Ingrese su opción (SI/NO)";
- 4 Leer opcion;
- 5 Si (opcion == "si" | opcion == "SI" | opcion == "Si" | opcion == "sI") Entonces
- 6 Escribir "Pusiste que sí";
- 7 SiNo
- 8 Escribir "Pusiste otra respuesta";



## Operador AND

El operador AND, también llamado Y o conjunción lógica, trabaja con dos operandos. Lo que hace este operador es devolver un valor VERDADERO si todos los operandos resultan VERDADERO, en caso contrario devuelve FALSO. Se representa con el símbolo &.

Seguiremos con el mismo ejemplo anterior, pero esta vez cambiaremos las reglas. Observemos como cambiar un o por un y hace una gran diferencia. La condición para pasar a cierto evento es ser mayor de edad y tener un nombre cuya longitud no supere los seis caracteres. Haremos un programa que permita al usuario ingresar su nombre y su edad. Luego la máquina informará si puede pasar o no, según la condición planteada.

Nombre	Edad	Salida
"Carlos"	25	"Puede pasar"
"María"	14	"NO Puede pasar"
"Ricardo"	40	"NO Puede pasar"
"Daniela"	8	"NO Puede pasar"

Para formalizar, la tabla de verdad del operador AND es la siguiente:

Expresión booleana	Resultado
VERDADERO & VERDADERO	VERDADERO
VERDADERO & FALSO	FALSO
FALSO & VERDADERO	FALSO
FALSO & FALSO	FALSO

El código puede ser el siguiente:

- 1 Algoritmo and
- 2 Definir nombre Como Cadena;
- 3 Definir edad Como Entero;
- 4 Escribir "Ingresa tu nombre";
- 5 Leer nombre;
- 6 Escribir "Ingresa tu edad";
- 7 Leer edad;
- 8 Si ( Longitud(nombre) <= 6 & edad >= 18 ) Entonces
- 9 Escribir "Puede pasar";
- 10 SiNo
- 11 Escribir "NO Puede pasar";



- 12 FinSi
- 13 FinAlgoritmo

Miremos las siguientes pruebas:

Una buena aplicación práctica de este operador podría ser para pedirle al usuario un valor numérico perteneciente a cierto rango. Supongamos que el usuario debe ingresar una nota numérica entre 0 y 10. Podría tener tal regla en una sola condición. Observemos el siguiente código:

- 1 Algoritmo and2
- 2 Definir num Como Entero;
- 3 Escribir "Ingrese un número entre 0 y 10 (inclusive)";
- 4 Leer num;
- 5 Si (num >= 0 & num <= 10) Entonces
- 6 Escribir "Número válido";
- 7 SiNo
- 8 Escribir "Número NO válido";
- 9 FinSi
- 10 FinAlgoritmo



