



# ESTRUCTURAS DE CONTROL.

Cristian David Henao H.

<http://www.facebook.com/codejavu>

<http://codejavu.blogspot.com/>



# ¿QUÉ SON LAS ESTRUCTURAS DE CONTROL?

A la hora de crear pseudocódigos o diagramas de flujo se tienen una serie de estructuras y técnicas que permiten definir algoritmos de una manera más simple y organizada antes de dar el paso a un lenguaje de programación.

Estas estructuras se conocen como estructuras algorítmicas o estructuras de control y definen que tipo de proceso seguir dependiendo de la problemática a resolver.



# ESTRUCTURAS DE CONTROL

Como se mencionó estas estructuras permiten definir el flujo del sistema, controlan el camino que debe seguir el algoritmo y se dividen en:

- Estructuras secuenciales.
- Estructuras condicionales o de decisión.
  - Condicionales simples
  - Condicionales dobles
  - Condicionales múltiples
- Estructuras iterativas o ciclos



# ESTRUCTURAS DE CONTROL

**Estructuras secuenciales:** Corresponde a los algoritmos donde el flujo del sistema se lee desde el inicio hasta el final de una manera secuencial.

**Estructuras condicionales o de decisión:** corresponde a los algoritmos donde el flujo de estos depende del resultado de una condición en específico.

**Estructuras iterativas o ciclos:** corresponde a los algoritmos que permiten repetir sus procesos internos la cantidad de veces que una condición lo permita.

NOMBRE ESTRUCTURA CONDICIONAL	
seudocódigo	Diagrama de flujo
si	if
si – sino	if – else
según sea – caso	switch - case

NOMBRE ESTRUCTURA ITERATIVA	
seudocódigo	Diagrama de flujo
repita – mientras	do - while
mientras	while
para	for



# ESTRUCTURAS SECUENCIALES

Este tipo de estructuras se usan cuando un proceso o instrucción sigue o es seguido por otro proceso, esto se realiza de forma secuencial, así la salida de uno es la entrada de otro y así sucesivamente hasta el final del algoritmo.

Diagrama de Flujo	Pseudocódigo
<pre>graph TD; Entry(( )) --&gt; A1[Acción 1]; A1 --&gt; A2[Acción 2]; A2 --&gt; A3[Acción 3]; A3 --&gt; Exit((...));</pre>	<pre>INICIO     Acción 1     Acción 2     Acción 3     ... FINAL</pre>



# ESTRUCTURAS SECUENCIALES

Ej.: Algoritmo para sumar 2 números enteros.

Diagrama de Flujo	Pseudocódigo
<pre>graph TD; INICIO([INICIO]) --&gt; Input[/a,b,result/]; Input --&gt; Process[result=a+b]; Process --&gt; Output[result]; Output --&gt; FIN([FIN]);</pre>	<pre>INICIO   numérico a,b,result;   result=a+b;   imprimir result; FINAL</pre>



# ESTRUCTURAS CONDICIONALES

Las estructuras condicionales se utilizan para tomar decisiones lógicas de ahí que se conozcan también como estructuras de decisión.

En las estructuras de decisión, se evalúa una condición y dependiendo del resultado de la misma se define que camino o acción realizar.

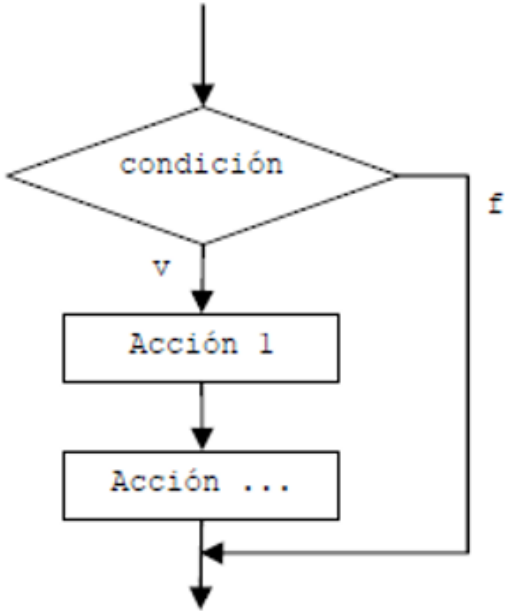
Se tienen 3 grupos de condiciones o estructuras selectivas, estos son:

- Condiciones Simples.
- Condiciones Dobles.
- Condiciones Múltiples.

# CONDICIONALES SIMPLES

Este tipo de estructuras permite ejecutar una determinada acción cuando se cumple determinada condición.

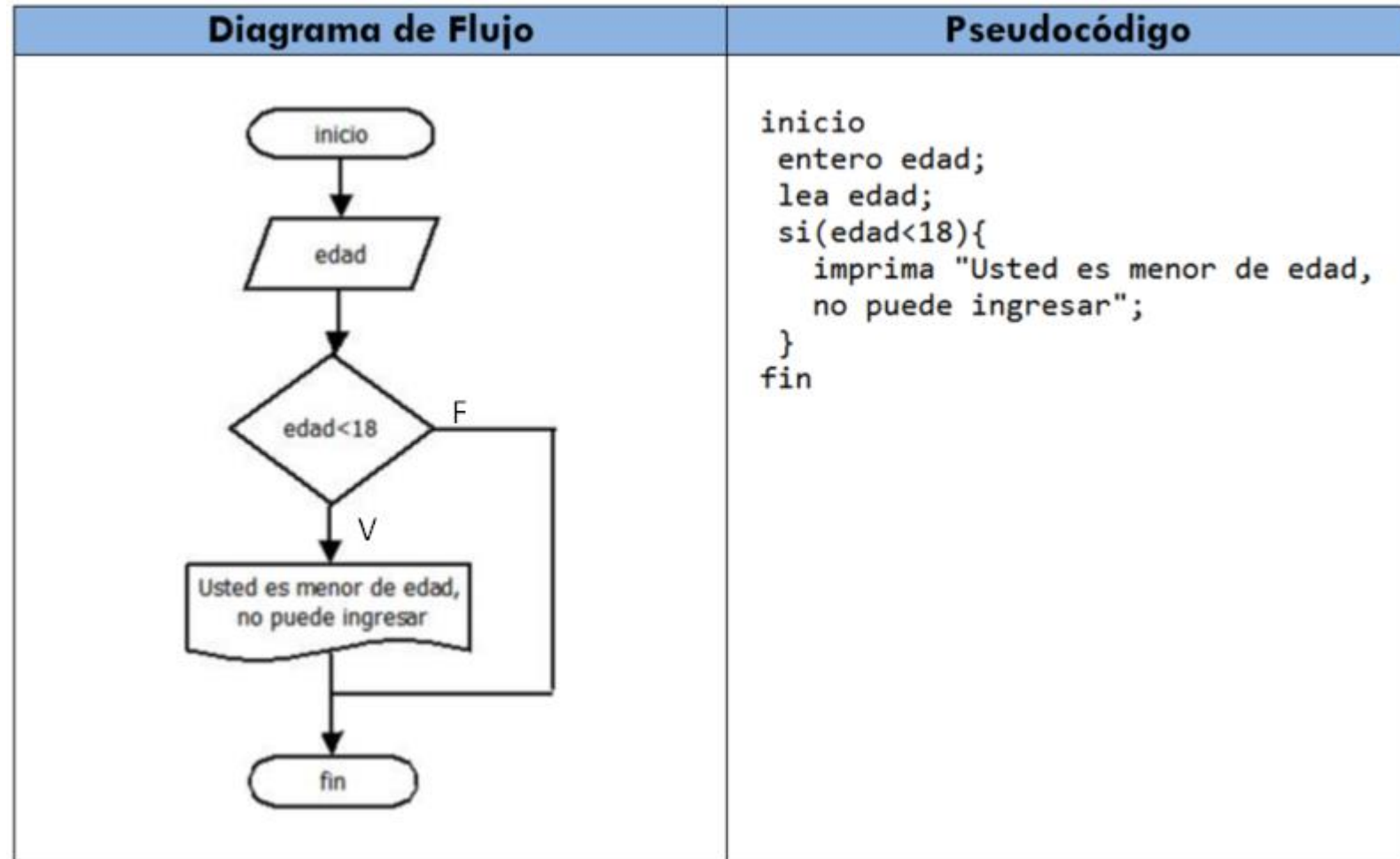
Para evaluar condiciones se usa la palabra SI, donde si el resultado de la condición es verdadera, se ejecutan las instrucciones dentro del condicional, si caso contrario el resultado es falso, entonces no se hace nada.

Diagrama de Flujo	Pseudocódigo
	<pre>... SI ( condicion ) {     Acción 1     Acción ... } ...</pre>



# CONDICIONALES SIMPLES

Ej.: Algoritmo que valide la entrada a una discoteca, si una persona menor de 18 años intenta ingresar el sistema debe presentar un mensaje de advertencia.





# CONDICIONALES DOBLES

Funciona de la misma manera que el anterior, solo que estas estructuras permiten no solo ejecutar una determinada acción cuando se cumple determinada condición, sino que en caso de que la condición no se cumpla, brinda otro camino para realizar acciones diferentes.

Si la condición es Verdadera, ejecuta acciones dentro del condicional

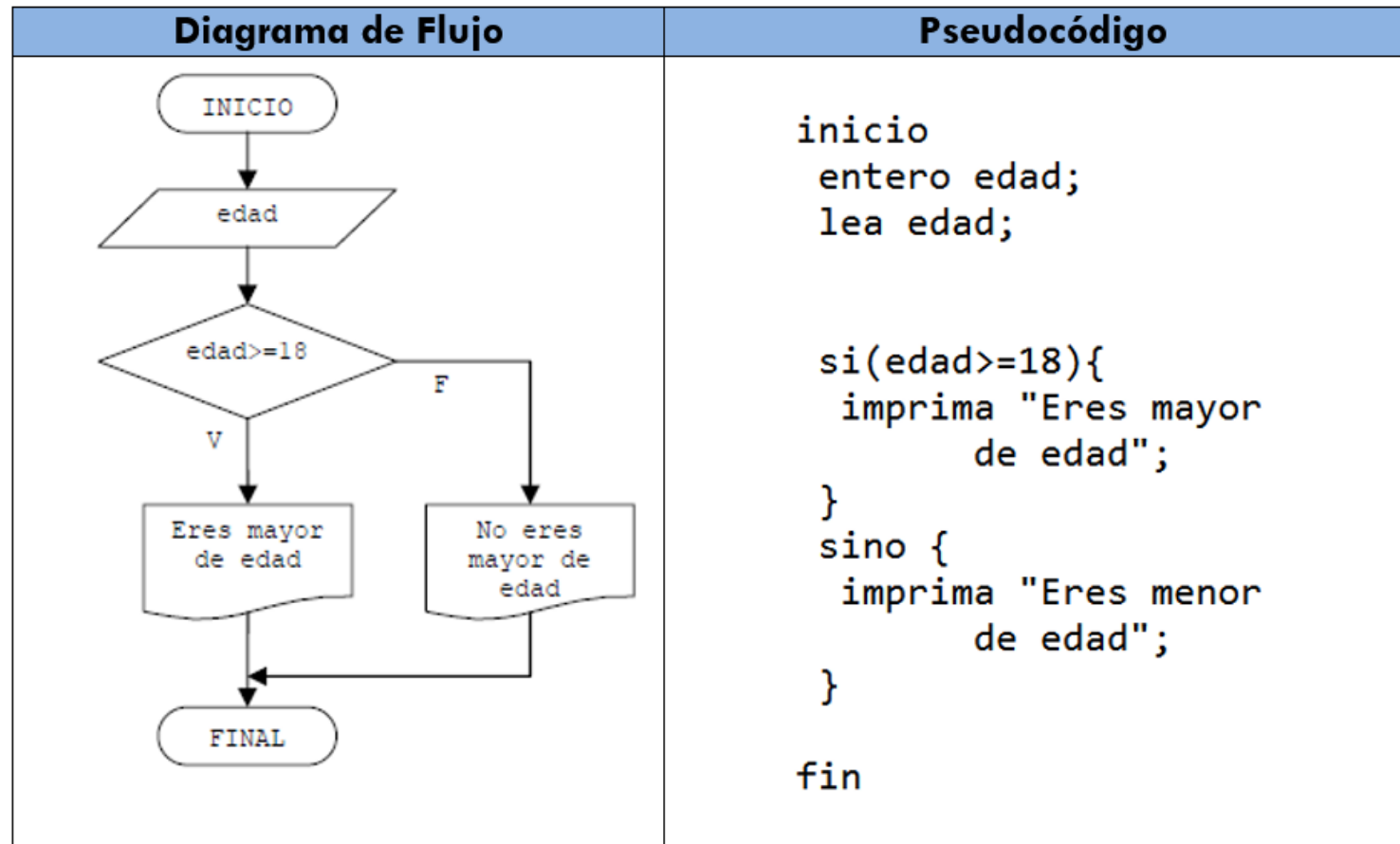
Si la condición es Falsa, ejecuta acciones dentro del SINO

Diagrama de Flujo	Pseudocódigo
<pre>graph TD; Entry(( )) --&gt; Cond{condición}; Cond -- v --&gt; Acc1[Acción 1]; Acc1 --&gt; Acc1_2[Acción ...]; Cond -- f --&gt; Acca[Acción a]; Acca --&gt; Acca_2[Acción ...]; Acc1_2 --&gt; Exit(( )); Acca_2 --&gt; Exit;</pre>	<pre>... SI ( condición ) {     Acción 1     Acción ... } SINO {     Acción a     Acción ... } ...</pre>



# CONDICIONALES DOBLES

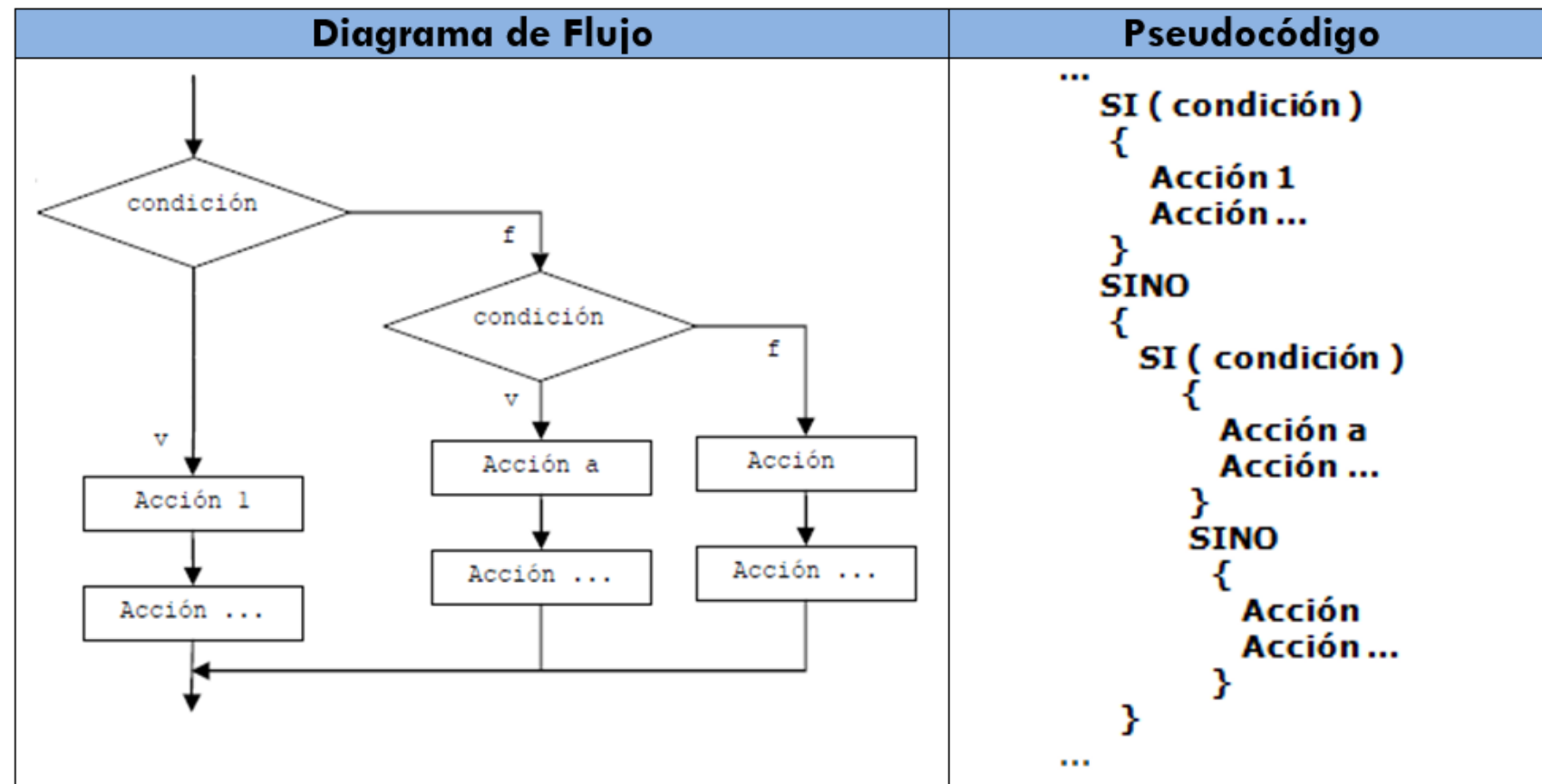
Ej.: Algoritmo que valide la edad de una persona, el sistema debe indicar si es menor o mayor de edad.



# CONDICIONALES MÚLTIPLES

Cuando se trabaja con condicionales, es muy común encontrar casos en los que después de tomar una decisión, se requiera seguir un posible camino donde se tengan que implementar nuevas condiciones, para esto se aplican las estructuras condicionales, donde en cada bloque de SI o SINO, pueden existir nuevas condiciones y dentro de estas nuevos procesos y así sucesivamente.

Hay que tener en cuenta que si el algoritmo usa muchas condiciones anidadas, puede tornarse complejo y extenso.





# CONDICIONALES MÚLTIPLES

Ej.: Algoritmo que determine si un número es positivo, negativo o cero.

Diagrama de Flujo	Pseudocódigo
<pre>graph TD     INICIO([INICIO]) --&gt; numero[/numero/]     numero --&gt; D1{numero &lt; 0}     D1 -- V --&gt; Negativo[Negativo]     D1 -- F --&gt; D2{numero &gt; 0}     D2 -- V --&gt; Positivo[Positivo]     D2 -- F --&gt; Cero[Cero]     Negativo --&gt; FINAL([FINAL])     Positivo --&gt; FINAL     Cero --&gt; FINAL</pre>	<pre>inicio entero numero; lea numero; si(numero&lt;0) {     imprima "Negativo"; } sino {     si(numero&gt;0)     {         imprima "Positivo";     }     sino     {         imprima "Cero";     } } fin</pre>



# CONDICIONALES MÚLTIPLES – CON BIFURCACIÓN

Cuando se requieren varias condiciones ya vimos que con decisiones anidadas se puede dar solución, sin embargo cuando se tiene un gran numero de condiciones se puede tornar más complejo, para dar solución a esto podemos usar la bifurcación.

Este tipo de estructuras permite escoger un camino a seguir dependiendo de una condición (que en este caso se comporta diferente a las condiciones ya vistas) que recibe la estructura **Según Sea**, la condición se evalúa y dependiendo de su contenido se escoge el flujo con las acciones correspondientes entre la cantidad de caminos posibles.

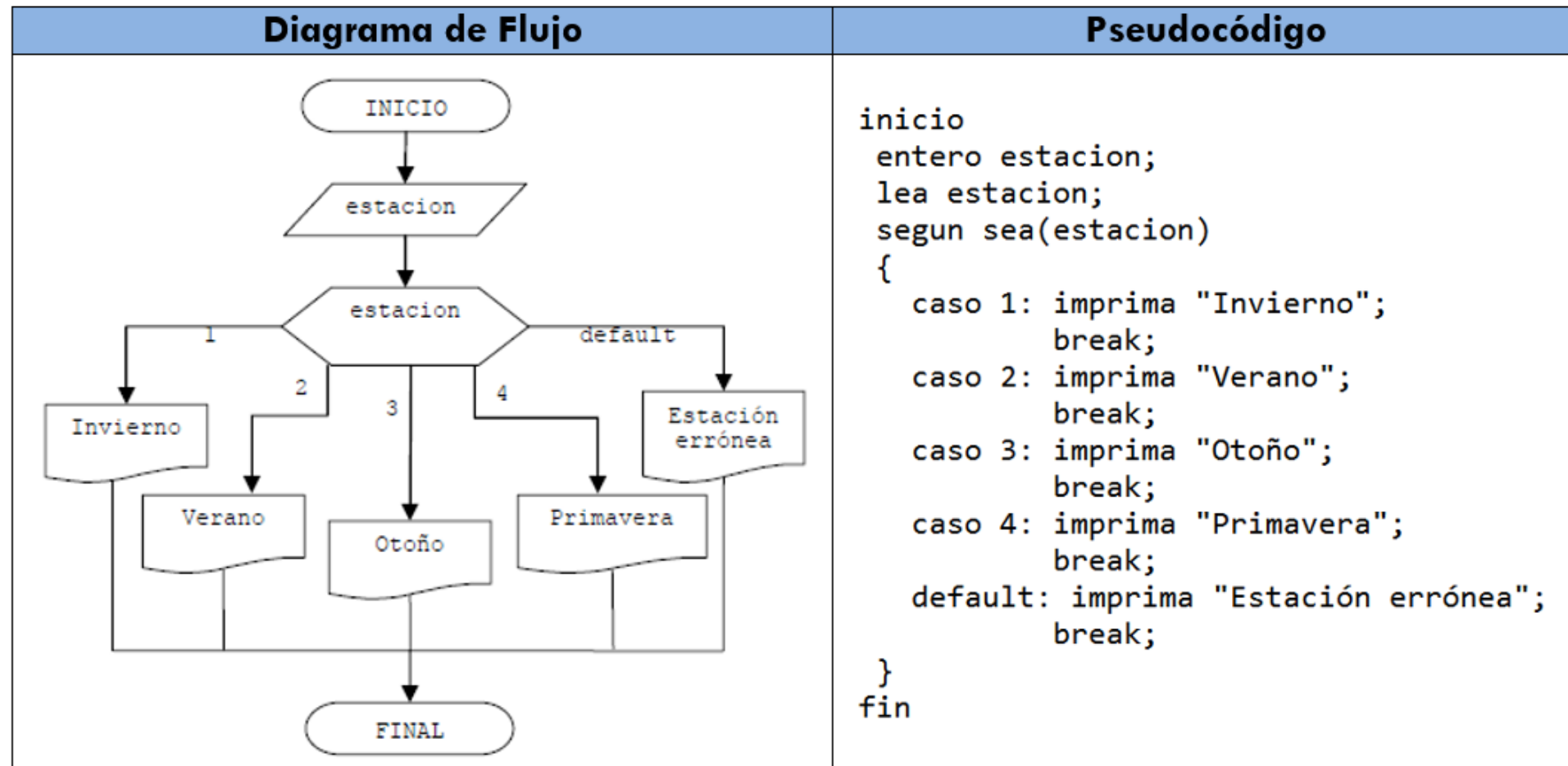
Diagrama de Flujo	Pseudocódigo
<pre>graph TD; Cond{condición} -- a --&gt; A1[Acción 1]; Cond -- b --&gt; A2[Acción 2]; Cond -- c --&gt; An1[Acción n]; Cond -- default --&gt; An2[Acción n]; A1 --&gt; Exit(( )); A2 --&gt; Exit; An1 --&gt; Exit; An2 --&gt; Exit;</pre>	<pre>SEGÚN SEA(condicion) {     CASO a:         Acción 1     CASO b:         Acción 2     CASO c:         Acción 3     DEFAULT:         Acción n }</pre>

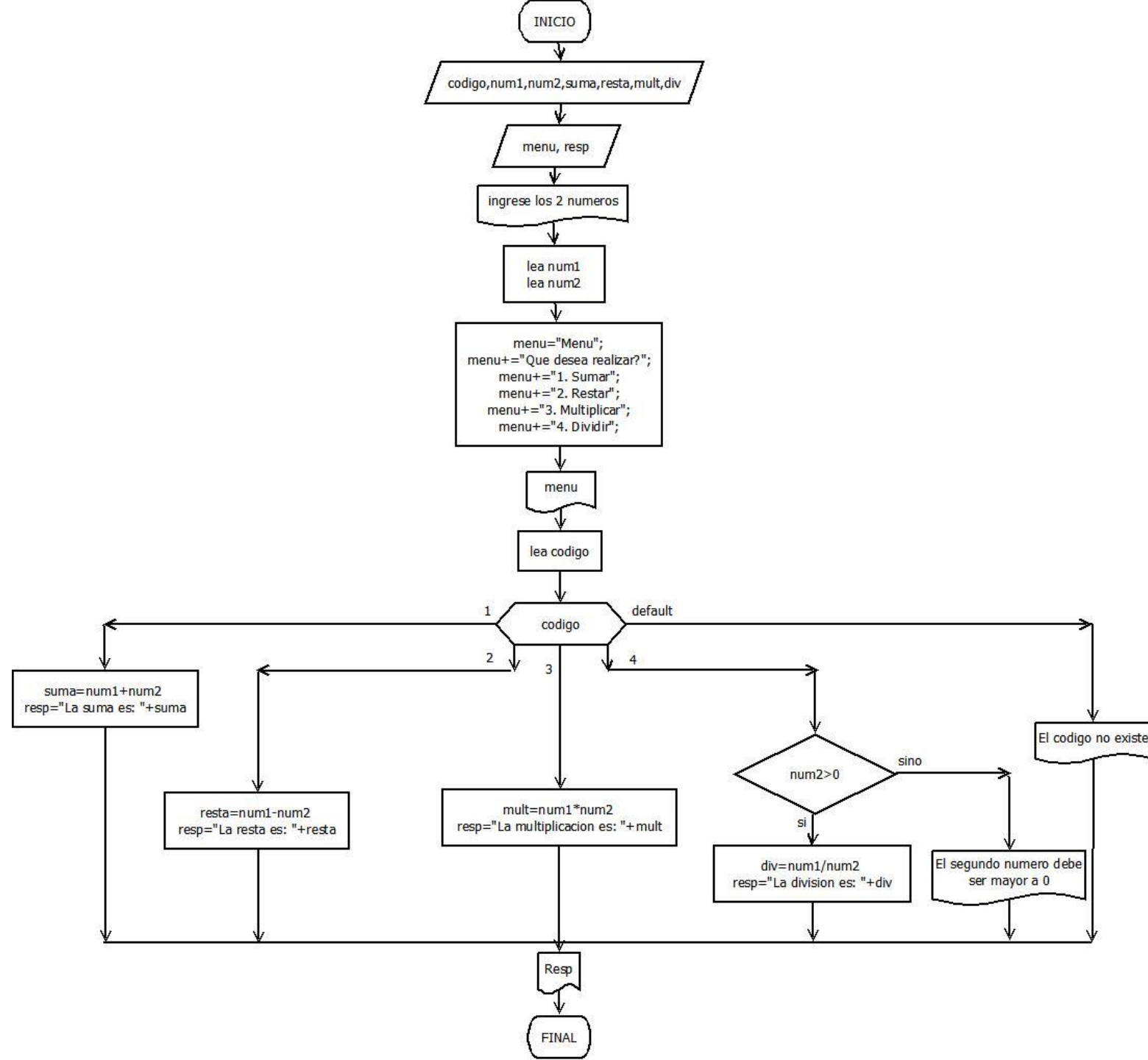


# CONDICIONALES MÚLTIPLES – CON BIFURCACIÓN

Ej.: Algoritmo que basado en la siguiente tabla permita leer un número e imprima el nombre de la estación correspondiente.

Código	Estación
1	Invierno
2	Verano
3	Otoño
4	Primavera









# ESTRUCTURAS ITERATIVAS

Las estructuras de iteración son todos los procesos que pueden ser repetitivos, en la vida diaria realizamos tareas repetitivas, por ejemplo, nos levantamos nos organizamos para irnos a trabajar, al terminar nuestra jornada regresamos a nuestras casas, descansamos y al otro día se repite el proceso, esto por lo regular durante toda la semana laboral.

Para poder modelar estos casos usando lógica de algoritmos, se tienen 3 estructuras principales que permiten dar soluciones repetitivas, a estos se le llaman ciclos y se conocen como.

- REPITA-MIENTRAS
- MIENTRAS
- PARA



# ESTRUCTURAS ITERATIVAS

La vida de un ciclo depende de una condición que indique si el ciclo se repite o si el ciclo se detiene.

**NOTA:** Toda estructura de iteración, debe tener una condición de parada para que finalice, de no ser así se puede encontrar con un ciclo infinito que impida el correcto funcionamiento de la aplicación.

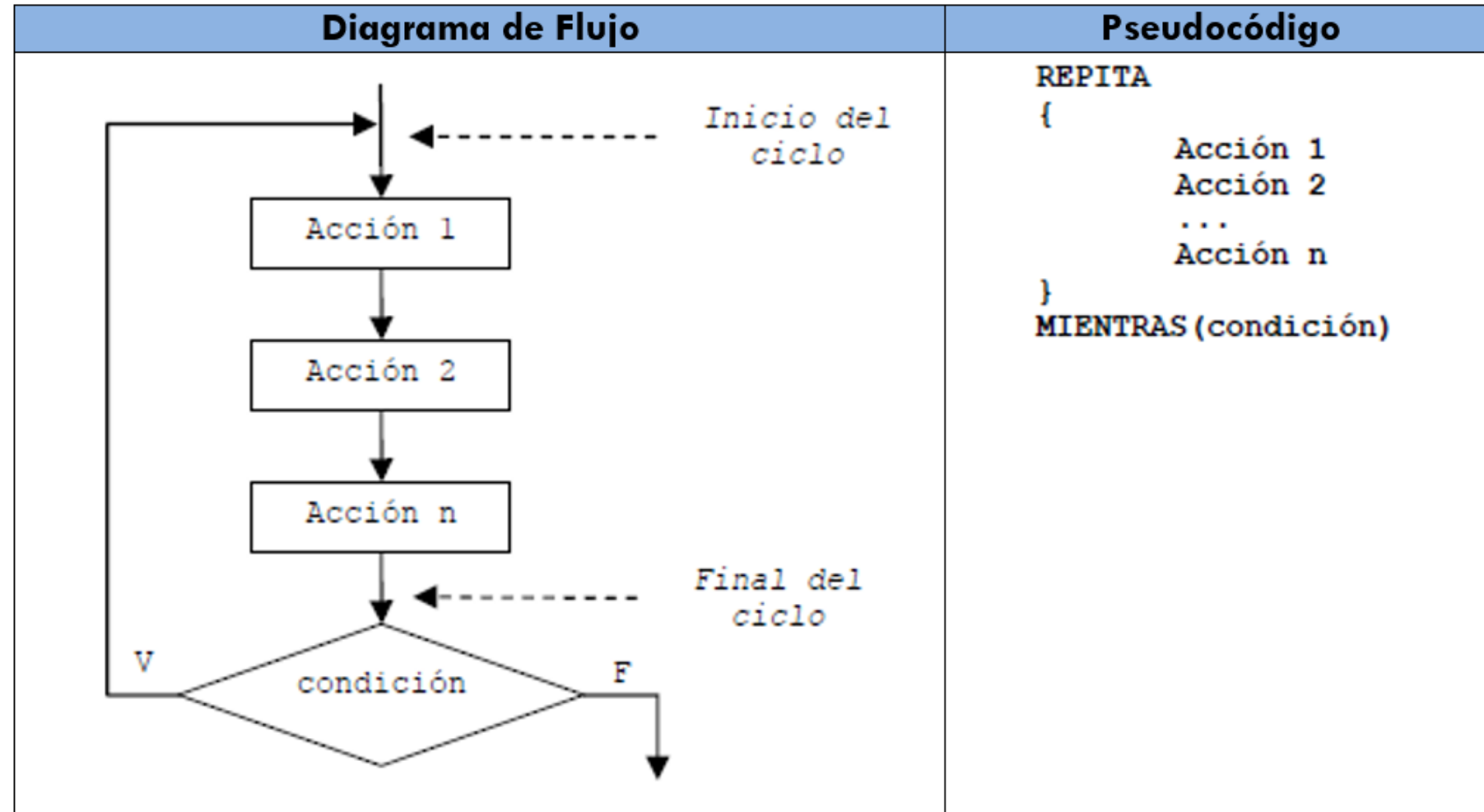
Basado en lo anterior es muy importante tener presente lo siguiente:

- Un ciclo se repite, solo si la condición es VERDADERA
- Un ciclo se termina (no se vuelve a repetir) si la condición es FALSA
- Debemos asegurarnos que en algún momento la condición debe ser falsa para que no se tenga un ciclo infinito.



# CICLO REPITA - MIENTRAS

En este ciclo primero se realizan una serie de acciones y al finalizar se evalúa una condición que dependiendo si es verdadera o falsa, realiza otra iteración repitiendo los procesos que queremos o simplemente rompe el ciclo y continúa con la secuencia de acciones que hay después de él.





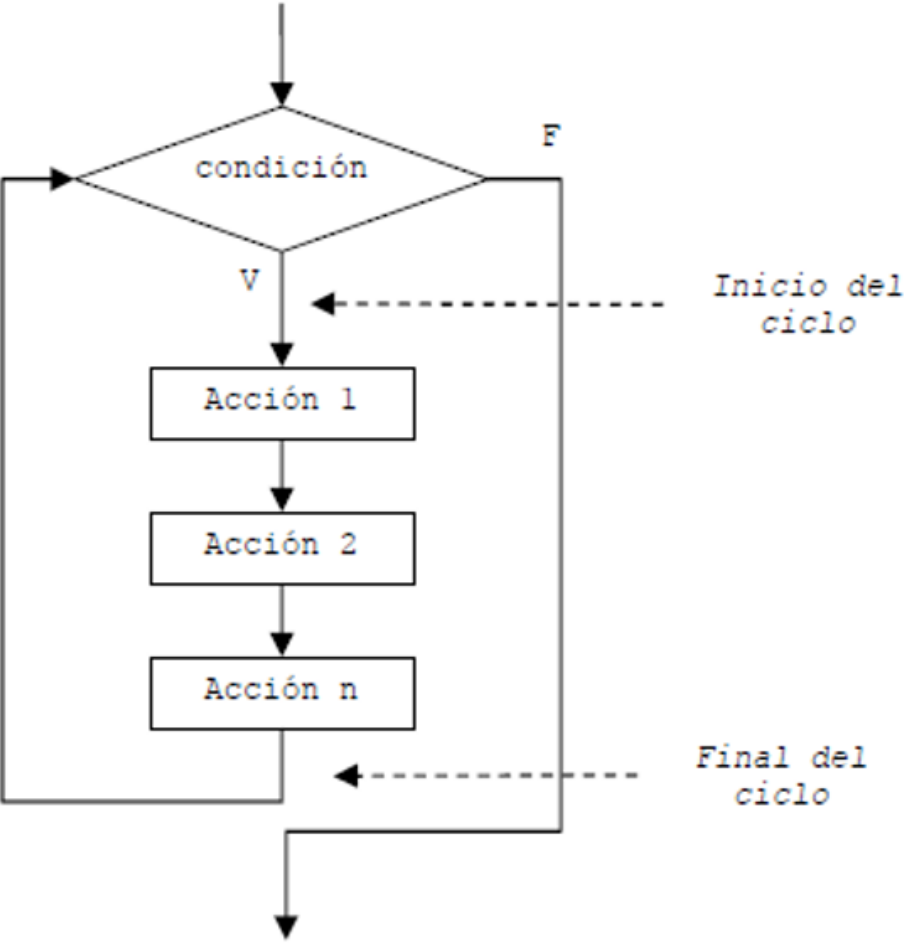
# CICLO REPITA - MIENTRAS

Ej.: Algoritmo que imprima los números del 1 al 10

Diagrama de Flujo	Pseudocódigo
<pre>graph TD; INICIO([INICIO]) --&gt; i1[i=1]; i1 --&gt; print[/i/]; print --&gt; iinc[i++]; iinc --&gt; cond{i &lt;= 10}; cond -- V --&gt; print; cond -- F --&gt; FINAL([FINAL]);</pre>	<pre>inicio   entero i=1;    repita   {     imprima "i";     i++;   }   mientras(i&lt;=10); fin</pre>

# CICLO MIENTRAS

A diferencia del ciclo REPITA-MIENTRAS, este ciclo primero realiza una validación, y dependiendo del resultado (Verdadero o Falso) determina si se ingresa o no al ciclo, repitiendo hasta que la condición falle.

Diagrama de Flujo	Pseudocódigo
	<pre>MIENTRAS (condición) {     Acción 1     Acción 2     ...     Acción n }</pre>



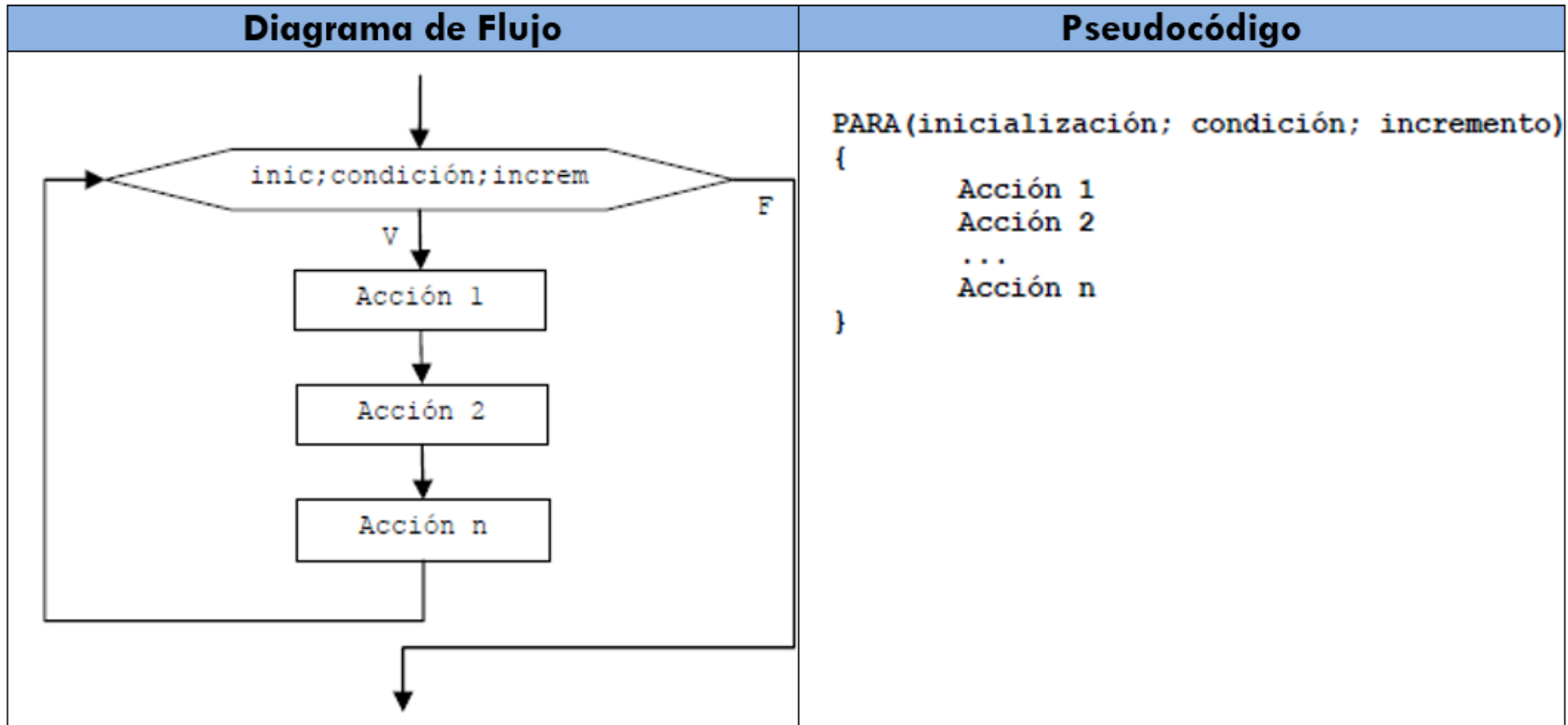
# CICLO MIENTRAS

Ej.: Algoritmo que imprima los números del 1 al 10

Diagrama de Flujo	Pseudocódigo
<pre>graph TD; INICIO([INICIO]) --&gt; i1[i=1]; i1 --&gt; Cond{i &lt;= 10}; Cond -- V --&gt; Print[/i/]; Print --&gt; Inc[i++]; Inc --&gt; Cond; Cond -- F --&gt; FIN([FIN]);</pre>	<pre>inicio entero i=1;  mientras(i&lt;=10) {     imprima "i";     i++; } fin</pre>

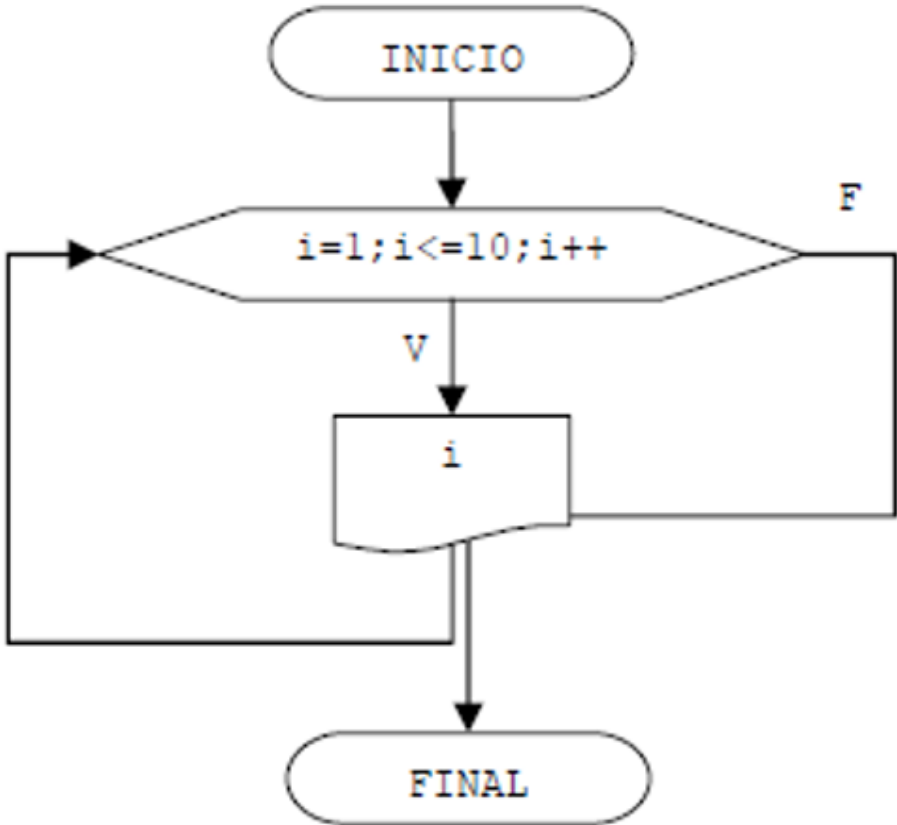
# CICLO PARA

Podemos decir que el ciclo PARA es la abreviación de los 2 ciclos anteriores, pues adicional a la condición que caracteriza los ciclos, también incluye la inicialización y el incremento



# CICLO PARA

Ej.: Algoritmo que imprima los números del 1 al 10

Diagrama de Flujo	Pseudocódigo
 <pre>graph TD; INICIO([INICIO]) --&gt; DECISION{i=1;i&lt;=10;i++}; DECISION -- V --&gt; PRINT[i]; PRINT --&gt; DECISION; DECISION -- F --&gt; FINAL([FINAL]);</pre> <p>The flowchart illustrates a loop process. It begins with an oval labeled 'INICIO'. An arrow points down to a diamond-shaped decision box containing the expression 'i=1;i&lt;=10;i++'. From the bottom of the diamond, an arrow labeled 'V' (Verdadero/True) points down to a rectangular process box labeled 'i'. From the bottom of the process box, an arrow points down to another diamond-shaped decision box. From the left side of this second diamond, an arrow loops back to the left side of the first diamond. From the right side of the second diamond, an arrow labeled 'F' (Falso/False) points right to an oval labeled 'FINAL'.</p>	<pre>inicio entero i;  para(i=0; i&lt;=10;i++) {     imprima "i"; } fin</pre>





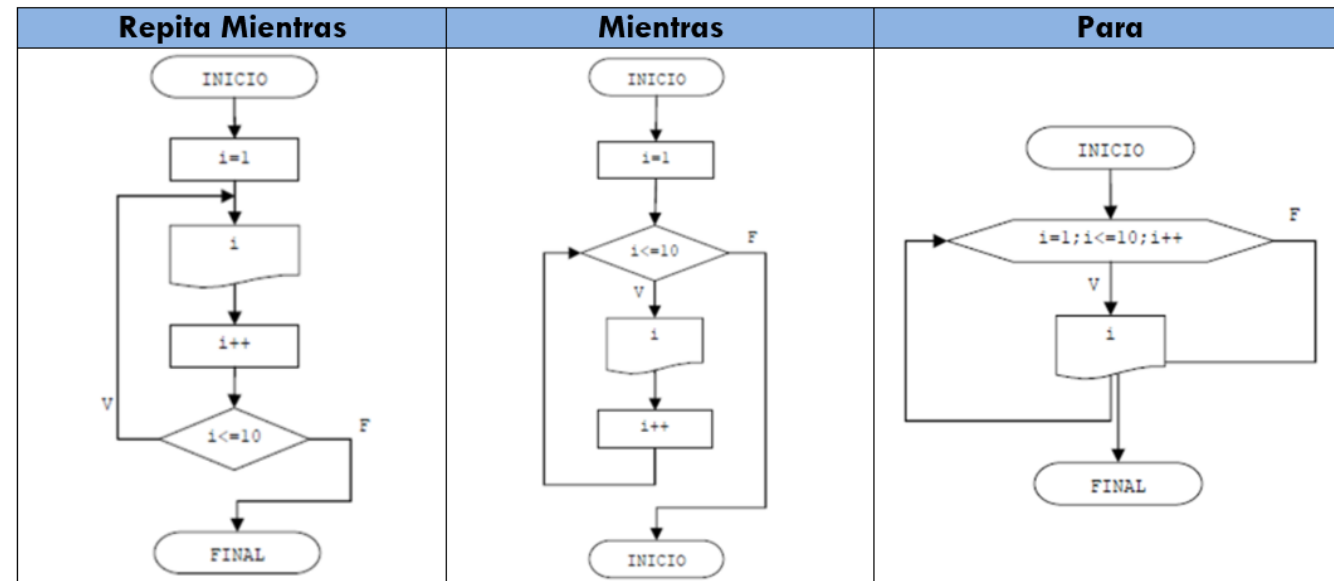
# COMPORTAMIENTO DE LOS CICLOS

Como vimos los 3 tipos de ciclos cumplen la misma función, sin embargo no todos los problemas se pueden resolver usando la misma estructura fácilmente, pues la lógica del ciclo PARA Y el MIENTRAS son diferente a la del ciclo REPITA-MIENTRAS.

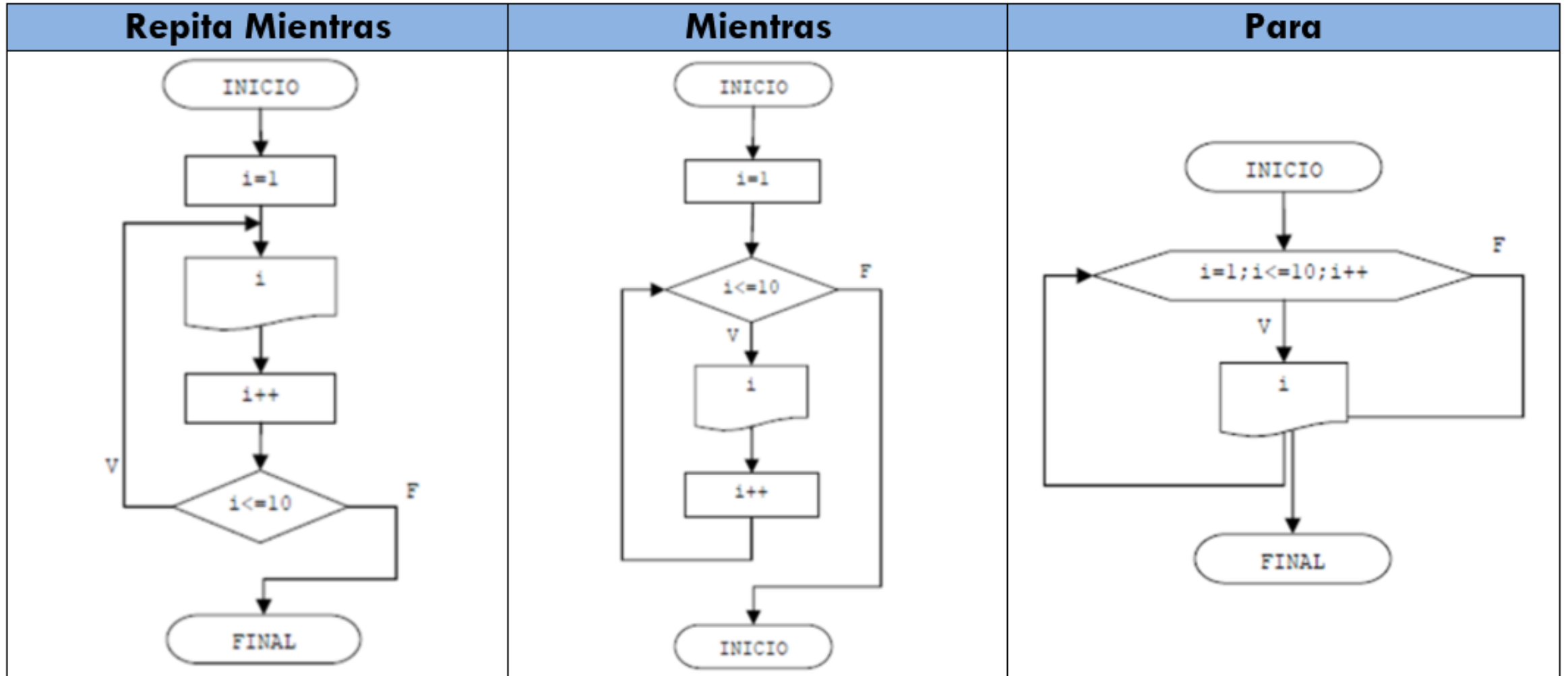
**REPITA-MIENTRAS:** Tiene una lógica Ejecución – Condición

**MIENTRAS:** Tiene una lógica Condición – Ejecución

**PARA:** Tiene una lógica Condición – Ejecución



# COMPORTAMIENTO DE LOS CICLOS



# COMPORTAMIENTO DE LOS CICLOS

Repita Mientras	Mientras	Para
<pre>inicio entero i=1;  repita {   imprima "i";   i++; } mientras(i&lt;=10); fin</pre>	<pre>inicio entero i=1;  mientras(i&lt;=10) {   imprima "i";   i++; } fin</pre>	<pre>inicio entero i;  para(i=1; i&lt;=10;i++) {   imprima "i"; } fin</pre>

