5. ESTRUCTURAS DE REPETICIÓN

Las estructuras de repetición, permiten la ejecución de una lista o secuencia de instrucciones (<bloque de instrucciones>) en varias ocasiones. El número de veces que el bloque de instrucciones se ejecutará se puede especificar de manera explícita, o a través de una condición lógica que indica cuándo se ejecuta de nuevo o cuándo no. A cada ejecución del bloque de instrucciones se le conoce como una **iteración**.

Existen cuatro tipos principales de sentencias de repetición:

- **Ciclo mientras**
- **Ciclo haga-mientras**
- **Ciclo repita-hasta**
- **Ciclo para**

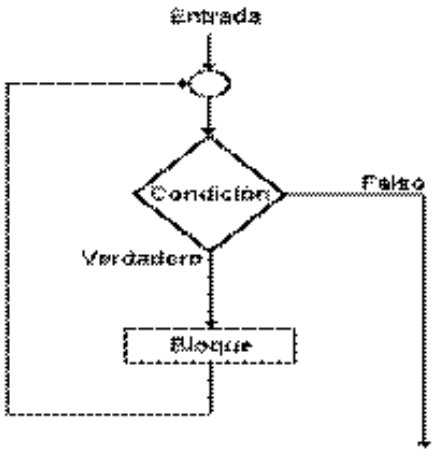
A continuación se describe cada una de ellas.

5.1 CICLO MIENTRAS

El **ciclo mientras** permite ejecuta un bloque de instrucciones *mientras* que una expresión lógica dada se cumpla, es decir, mientras su evaluación dé como resultado *verdadero*. La expresión lógica se denomina *condición* y siempre se evalúa antes de ejecutar el bloque de instrucciones. Si la condición no se cumple, el bloque no se ejecuta. Si la condición se cumple, el bloque se ejecuta, después de lo cual la instrucción vuelve a empezar, es decir, la condición se vuelve a evaluar.

En el caso en que la condición evalúe la primera vez como falsa, el bloque de instrucciones no será ejecutado, lo cual quiere decir que el número de repeticiones o ***iteraciones*** de este bloque será cero. Si la condición siempre evalúa a verdadero, la instrucción se ejecutará indefinidamente, es decir, un número infinito de veces.

La forma general del ciclo mientras es la siguiente:

DIAGRAMA DE FLUJO	PSEUDOCODIGO
	**mientras** <condición> **hacer** <bloque instrucciones> **fin_mientras**

Donde, **<condición>** es la expresión lógica que se evalúa para determinar la ejecución o no

del bloque de instrucciones, y **<bloque instrucciones>** es el conjunto de instrucciones que se ejecuta si la condición evalúa a Verdadero.

Ejemplos.

Ejemplo 1. Dado un número natural n se desea calcular la suma de los números naturales desde 1 hasta n.

ANALISIS DEL PROBLEMA:

Variables Conocidas	Un número natural.
Variables Desconocidas	Un número natural.
Condiciones	El número buscado es la suma de los naturales empezando en cero hasta el número dado.

ESPECIFICACIÓN:

Entradas	n ∈ Enteros, n ≥ 0 (n es el número dado).
Salidas	suma ∈ Enteros, suma ≥ 0 suma es la sumatoria de los primeros n números naturales.

DISEÑO:

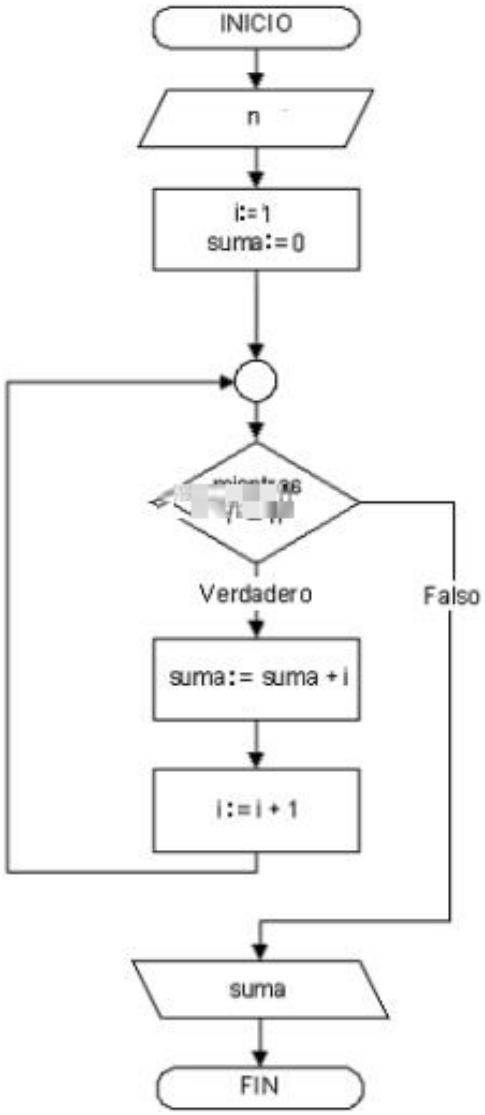
Primera División:

Inicio
Paso 1. Leer el número.
Paso 2. Recorrer los números desde cero hasta el número dado e irlos sumando.
Paso 3. Imprimir la suma
Fin

División Final:

```
1  n: entero /* se define la variable para el número */
2  suma: entero /* se define la variable para la suma */
3  i: entero /* se define la variable para recorrer los números entre 0 y n */

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

7  i :=1 /* empieza la variable que recorre los números en 0 */

8
9  mientras (i <= n) hacer
10    suma := suma + i /* en cada iteración suma el número i */
      i := i + 1 /* para tomar el siguiente número en la próxima iteración */

11
12 fin_mientras
   escribir  ("La suma es: ", suma)
```

Diagrama de Flujo:



PRUEBA DE ESCRITORIO:

Este algoritmo cuenta con doce (12) líneas, las tres primeras, son para definir las variables usadas y las últimas nueve son las instrucciones que son aplicadas sobre dichos datos. De esta manera la prueba de escritorio se debe realizar solamente sobre las líneas 4-12, teniendo en cuenta los valores para las variables.

LINEA	n	i	suma	ENTRADA	SALIDA
4					Ingrese el número:
5	5			5	
6			0		
7		0			
8	La condición es evaluada a verdadero, por lo tanto se ejecuta el bloque de acciones del ciclo, es decir, pasa a la línea 9.				
9			0		
10		1			
11	Se salta hasta la línea que contiene la condición del ciclo mientras en ejecución, es decir, hasta la línea 8				
8	La condición es evaluada a verdadero, por lo tanto se ejecuta el bloque de acciones del ciclo, es decir, pasa a la línea 9.				
9			1		
10		2			
11	Se salta hasta la línea que contiene la condición del ciclo mientras en ejecución, es decir, hasta la línea 8				
8	La condición es evaluada a verdadero, por lo tanto se ejecuta el bloque de acciones del ciclo, es decir, pasa a la línea 9.				
9			3		
10		3			
11	Se salta hasta la línea que contiene la condición del ciclo mientras en ejecución, es decir, hasta la línea 8				
8	La condición es evaluada a verdadero, por lo tanto se ejecuta el bloque de acciones del ciclo, es decir, pasa a la línea 9.				
9			6		
10		4			
11	Se salta hasta la línea que contiene la condición del ciclo mientras en ejecución, es decir, hasta la línea 8				
8	La condición es evaluada a verdadero, por lo tanto se ejecuta el bloque de acciones del ciclo, es decir, pasa a la línea 9.				
9			10		
10		5			
11	Se salta hasta la línea que contiene la condición del ciclo mientras en ejecución, es decir, hasta la línea 8				

8	La condición es evaluada a verdadero, por lo tanto se ejecuta el bloque de acciones del ciclo, es decir, pasa a la línea 9.				
9			15		
10		6			
11	Se salta hasta la línea que contiene la condición del ciclo mientras en ejecución, es decir, hasta la línea 8				
8	La condición evalúa a falso, por lo tanto no se ejecuta el bloque de acciones del ciclo y este termina, es decir, pasa a la línea 12, la línea siguiente a la línea del fin_mientras del ciclo.				
12					La suma es: 15

Ejemplo 2. Calcular el máximo común divisor de dos números naturales, distintos de cero.

ANALISIS DEL PROBLEMA:

Variables Conocidas	dos números naturales.
Variables Desconocidas	Un número natural.
Condiciones	El número buscado es el máximo común divisor de los números conocidos.

ESPECIFICACIÓN:

Entradas	a, b $\in$ Enteros, (a, b son los números dados).
Salidas	mcd $\in$ Enteros, (mcd es el máximo común divisor de los números dados).
Condiciones	mcd = max { 1 <= k <= min{a,b} | a mod k = 0 | b mod k = 0}

DISEÑO:

Primera División:

Inicio
Paso 1. Leer los números.
Paso 2. Calcular el máximo común divisor de los números dados.
Paso 3. Imprimir el máximo común divisor
Fin

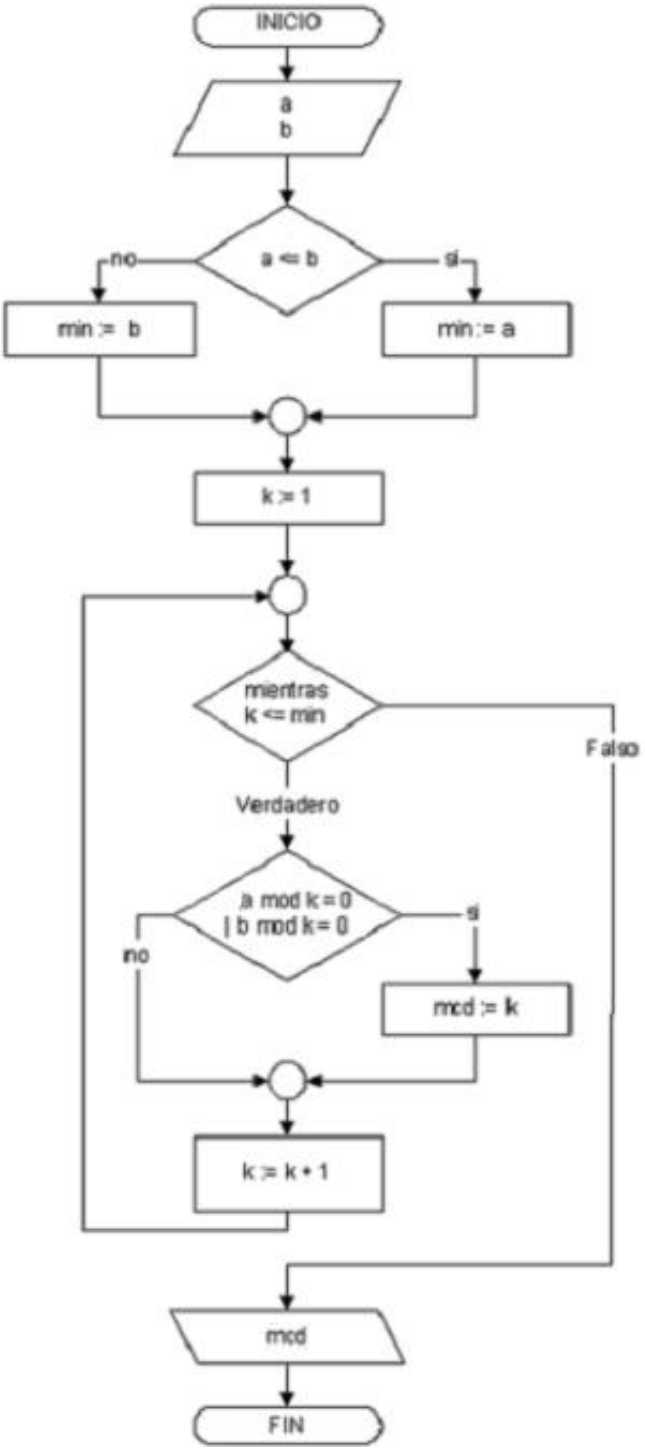
Segunda División:

Inicio
Paso 1. Leer los números.
Paso 1.1. Leer el primer número
Paso 1.2. Leer el segundo número
Paso 2. Calcular el máximo común divisor de los números dados.
Paso 2.1. Determinar el mínimo de los dos números dados.
Paso 2.2. Recorrer los números desde el uno hasta el mínimo de los dos números e ir determinando si cada número cumple la propiedad de dividir a los números dados. El número mayor es el máximo común divisor.
Paso 3. Imprimir el máximo común divisor.
Fin

División Final:

```
a: entero /* se define la variable para el primer número */
b: entero /* se define la variable para el segundo número */
mcd: entero /* se define la variable para el mcd */
min: entero /* se define la variable para el mínimo */
k: entero /* se define la variable para recorrer los números entre 1 y min */

escribir ( "Ingrese el primer número: " )
leer (a) /* lee el primer número */
escribir( "Ingrese el segundo número: " )
leer (b) /* lee el segundo número */

si (a < b)entonces /* se calcula el mínimo de los números */
   min := a
sino
  min := b
fin_si

k :=1 /* empieza en 1 la variable que recorre los posibles divisores */

mientras (k <= min)hacer
   si (a mod k = 0) & (b mod k = 0) entonces
      mcd := k
   fin_si

   k := k + 1 /*incrementa k en 1 para tomar el siguiente número en la próxima iteración */
fin_mientras

escribir ( "El máximo común divisor es: ", mcd )
```

Diagrama de Flujo:



PRUEBA DE ESCRITORIO:

Este algoritmo cuenta con veinticuatro (22) líneas, las cinco primeras, son para definir las variables usadas y las últimas diecisiete son las instrucciones que son aplicadas sobre dichos datos. De esta manera la prueba de escritorio se debe realizar solamente sobre las líneas 6-22, teniendo en cuenta los valores para las variables.

LINEA	a	b	Min	K	Mcd	ENTRADA	SALIDA
6							Ingrese el primer número:

7	8					8	
8							Ingrese el segundo número:
9		6				6	
10	La condición evalúa a falso, se pasa a la línea siguiente al sino, es decir, a la línea 13.						
11			6				
12	Se salta a la línea siguiente al fin_si, es decir, a la línea 15						
15				1			
16	La condición evalúa a verdadero (1<=6), luego se ejecuta el bloque de acciones del ciclo mientras.						
17	La condición evalúa a verdadero, se ejecuta la línea siguiente al entonces, línea 18.						
18					1		
19	Se salta a la siguiente línea al fin_si, línea 21.						
21				2			
22	Se retorna a la línea de inicio del ciclo mientras, línea 16						
16	La condición evalúa a verdadero (2<=6), luego se ejecuta el bloque de acciones del ciclo mientras.						
17	La condición evalúa a verdadero, se ejecuta la línea siguiente al entonces, línea 18.						
18					2		
19	Se salta a la siguiente línea al fin_si, línea 21.						
21				3			
22	Se retorna a la línea de inicio del ciclo mientras, línea 16						
16	La condición evalúa a verdadero (3<=6), luego se ejecuta el bloque de acciones del ciclo mientras.						
17	La condición evalúa a falso, se ejecuta la línea siguiente al sino, línea 20.						
21	Se salta a la siguiente línea al fin_si, línea 21.						
21				4			
22	Se retorna a la línea de inicio del ciclo mientras, línea 16						
16	La condición evalúa a verdadero (4<=6), luego se ejecuta el bloque de acciones del ciclo mientras.						
17	La condición evalúa a falso, se ejecuta la línea siguiente al sino, línea 20.						
21	Se salta a la siguiente línea al fin_si, línea 21.						
21				5			
22	Se retorna a la línea de inicio del ciclo mientras, línea 16						
16	La condición evalúa a verdadero (5<=6), luego se ejecuta el bloque de acciones del ciclo mientras.						
17	La condición evalúa a falso, se ejecuta la línea siguiente al sino, línea 20.						
21	Se salta a la siguiente línea al fin_si, línea 21.						
21				6			
22	Se retorna a la línea de inicio del ciclo mientras, línea 16						
16	La condición evalúa a verdadero (6<=6), luego se ejecuta el bloque de acciones del ciclo mientras.						
17	La condición evalúa a falso, se ejecuta la línea siguiente al sino, línea 20.						
21	Se salta a la siguiente línea al fin_si, línea 21.						
21				7			
22	Se retorna a la línea de inicio del ciclo mientras, línea 16						
16	La condición evalúa a verdadero (7<=6), luego no se ejecuta el bloque de acciones del ciclo mientras y se salta a la línea siguiente al fin_mientras, línea 23						

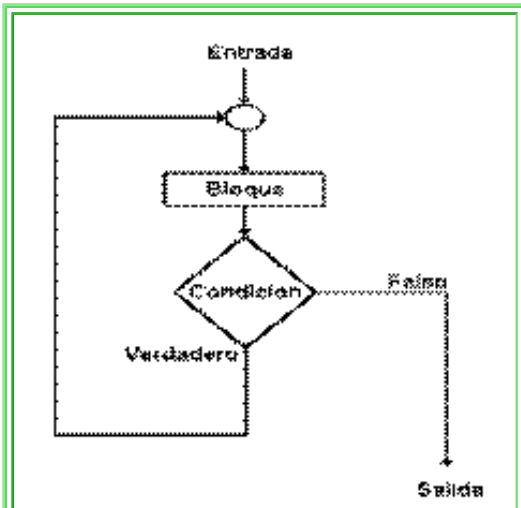
17	La condición evalúa a falso, se ejecuta la línea siguiente al sino, línea 20.						
23							El máximo común divisor es: 2

5.2 CICLO HAGA-MIENTRAS

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

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

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



haga
<bloque instrucciones>
mientras <condición>

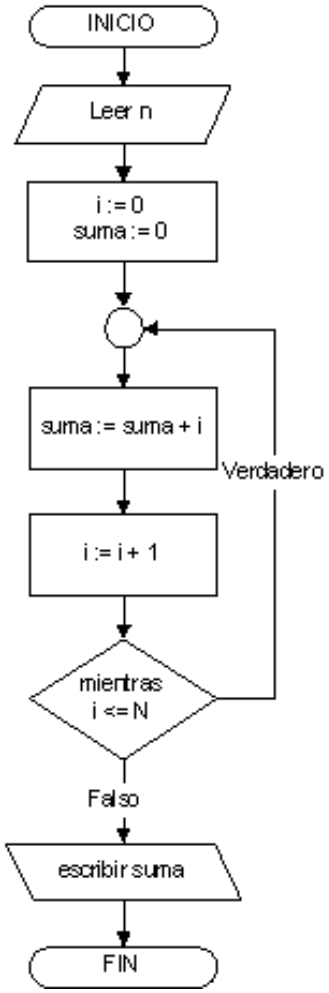
Donde, <bloque instrucciones> es el conjunto de instrucciones que se ejecuta y <condición> es la expresión lógica que determina si el bloque se ejecuta. Si la <condición> se evalúa como **verdadero** el bloque es ejecutado de nuevo y si es evaluada como **falso** no es ejecutado. Después de ejecutar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

Ejemplos

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

```
1  n: entero /* se define la variable para el número */
2  suma: entero /* se define la variable para la suma */
3  i: entero /* se define la variable para recorrer los números entre 0 y n */

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

7  i :=0 /* empieza la variable que recorre los números en 0 */

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

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

Diagrama de Flujo:



Ejemplo 2. Realizar un programa que le presente un menú al usuario con las siguientes opciones:

1. Leer dos números,
2. Sumar los dos números leídos,
3. Restarle al primer número el segundo,
4. Multiplicar los dos números,
5. Dividir el primer número dado por el segundo,
6. Imprimir el número resultado de la última operación realizada y 0. Para terminar.

Después de que el usuario determine la operación a realizar el programa debe realizarla. Se debe garantizar que el usuario haya ingresado los dos números antes de poder realizar cualquier operación y que no se puede imprimir un resultado sin la realización de una operación.

ANALISIS DEL PROBLEMA:

Variables Conocidas	La opción escogida por el usuario y dos números reales.
Variables Desconocidas	Un número real o un texto.

Condiciones	El número buscado es el resultado de la última operación realizada sobre los números dados. En el caso que no se ingresarán datos o no se hubiere realizado una operación, se debe presentar un mensaje de error.

ESPECIFICACIÓN:

Entradas	a, b ∈ Reales (a, b son los números dados), opción ∈ Enteros (opción es la operación deseada por el usuario).
Salidas	resp ∈ Reales, (resp es el resultado de la última operación realizada sobre los números dados, si está existiera) y texto ∈ Cadenas (texto es una cadena que indica que una operación no ha sido realizada, debido a que falta leer los datos o no se ha realizado operación alguna o el valor de la operación).
Condiciones	• **desconocido** si no se han leído a y b • **a + b** si a y b se han leído y opción es 2. • **resp = a - b** si a y b se han leído y opción es 3. • **a * b** si a y b se han leído y opción es 4. • **a / b** si a y b se han leído y opción es 5. • **"Datos no leídos"** si no se han leído a y b y opción es 2, 3, 4, 5 y 6 • **texto = "No se realizo operación"** si opción es 6 y resp es desconocido. • **"El resultado es:"**, resp. En otro caso.

DISEÑO:

Primera División:

Inicio
hacer
Paso 1. Presentar menú.
Paso 2. leer la opción.
Paso 3. Realizar la operación de acuerdo a la opción **mientras** opción sea diferente de terminar.
Fin

Segunda División:

Inicio
hacer
Paso 1. Presentar menú.
Paso 2. leer la opción.
Paso 3. seleccionar <opción> **de**
Paso 3.1. si es 1
Paso 3.1.1. leer los números
Paso 3.1.2. Salir
Paso 3.2. si es 2
Paso 3.2.1. Sumar los números si ya han sido leídos
Paso 3.2.2. Salir
Paso 3.3. si es 3
Paso 3.3.1. Restar los números si ya han sido leídos
Paso 3.3.2. Salir
Paso 3.4. si es 4
Paso 3.4.1. Multiplicar los números si ya han sido leídos

Paso 3.4.2. Salir
Paso 3.5. si es 5
Paso 3.5.1. Dividir los números si ya han sido leídos
Paso 3.5.2. Salir
Paso 3.6. si es 6
Paso 3.6.1. Imprimir el resultado de la última operación, si ya se ha realizado una operación
Paso 3.6.2. Salir
Paso 4. Realizar la operación de acuerdo a la opción **mientras** opción sea diferente de terminar.
Fin

División final:

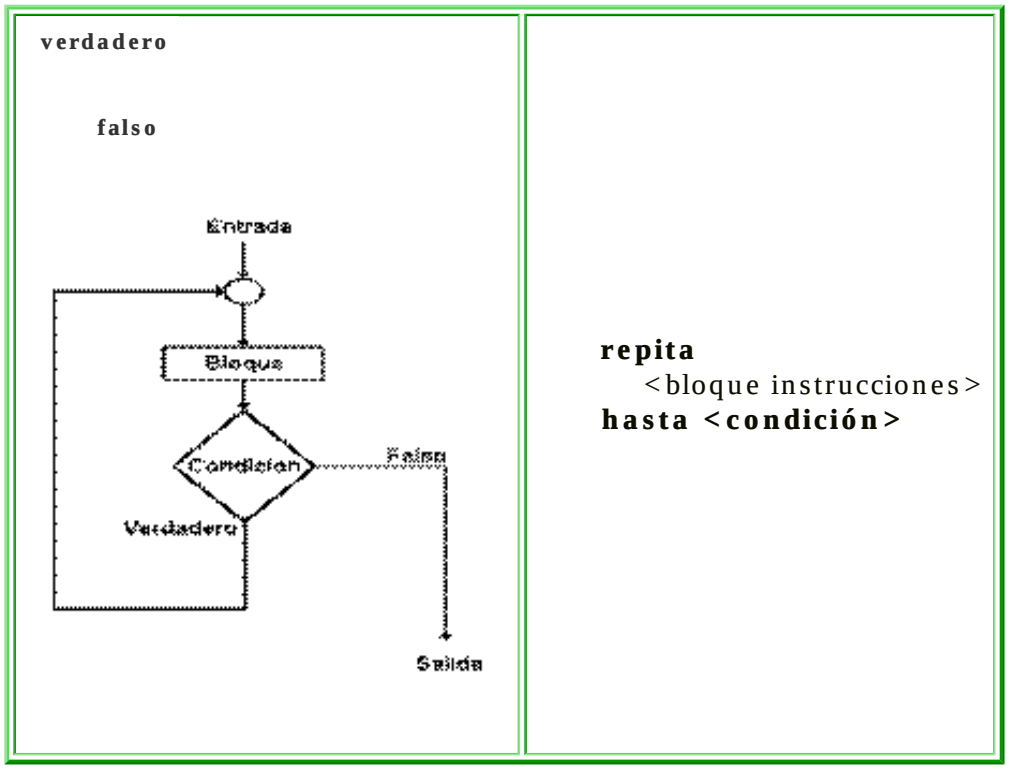
```
a, b, c: real
datos_leidos: booleano
operacion_realizada: booleano
datos_leidos := falso
operación_realizada :=falso
haga
  escribir ("Menú de operaciones" )
  escribir ( "0. Terminar" )
  escribir ( "1. Leer datos" )
  escribir ( "2. Sumar datos" )
  escribir ( "3. Restar datos" )
  escribir ( "4. Multiplicar datos" )
  escribir ( "5. Dividir datos" )
  escribir ( "6. Mostrar Resultado" )
  leer  (opcion)
  seleccionar ( opcion) de:
   caso 1:  escribir ("Ingrese el primer número")
         leer a
         escribir ("Ingrese el segundo número")
         leer b
         datos_leidos := verdadero
         operacion_realizada := falso
    caso 2:  si ( datos_leidos ) entonces
                      c := a + b
                      operacion_realizada :=verdadero
              sino
                   escribir ("Error. no se han leído los datos ****" )
              fin_si
   caso 3:  si ( datos_leidos ) entonces
               c := a - b
               operacion_realizada :=verdadero
             sino
               escribir ("Error. no se han leído los datos ****" )
             fin_si
   caso 4:  si ( datos_leidos ) entonces
               c := a * b
               operacion_realizada :=verdadero
             sino
               escribir ("Error. no se han leído los datos ****" )
            fin_si
   caso 5:  si ( datos_leidos ) entonces
                c := a / b
                operacion_realizada :=verdadero
```

```
            sino
                escribir ("Error. no se han leído los datos ****" )
            fin_si
    caso6:  si ( operacion_realizada ) entonces
                escribir ("El resultado de la última operación es:")
                escribir  c
            sino
                escribir ("Error. no se han leido los datos ****" )
            fin_si
fin_seleccionar
mientras (opcion <>0)
```

5.3 CICLO REPITA-HASTA

El **ciclo repita-hasta** es similar al **ciclo haga-mientras**, la diferencia está en que el bloque de instrucciones se ejecuta nuevamente si la condición es evaluada como **falso** y no se ejecuta más si es evaluada como **verdadero**. Sobra advertir que el bloque de instrucciones se ejecuta por lo menos una vez.

La forma general del ciclo repetir-hasta es la siguiente:



repita
    <bloque instrucciones>
hasta <condición>

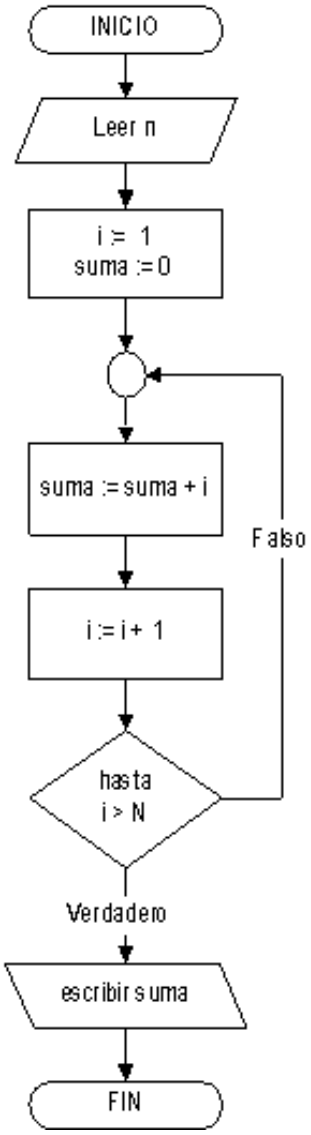
Donde, <**bloque instrucciones**> es el conjunto de instrucciones que se ejecuta y <**condición**> es la expresión lógica que determina si el bloque es ejecutado nuevamente. Si la <condición> se evalúa como **falso** el bloque es ejecutado de nuevo y si es evaluada como **verdadero** no es ejecutado.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

Ejemplos.

Ejemplo 1. El problema de calcular la suma de los números naturales desde 1 hasta n (enunciado anteriormente), se puede solucionar usando el **ciclo repita-hasta**. A continuación se describe el algoritmo solución:

```
n: entero /* se define la variable para el número */
suma: entero /* se define la variable para la suma */
i: entero /* se define la variable para recorrer los números entre 0 y n */

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

i :=0 /* empieza la variable que recorre los números en 0 */

repita
    suma := suma + i /* en cada iteración suma el número i */
    i := i + 1 /* incrementa i en 1 para tomar el siguiente número en la próxima iteración */
hasta (i > n)

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

Diagrama de Flujo:

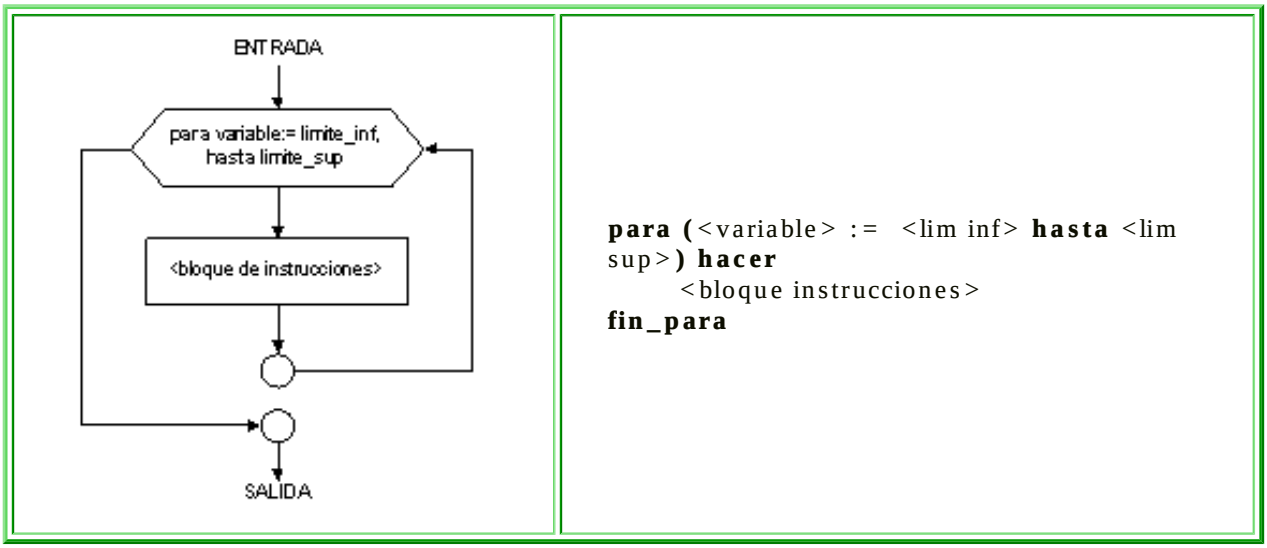


5.4 CICLO PARA

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

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

La forma general del ciclo para es la siguiente:



para (<variable> := <lim inf> **hasta** <lim sup>**) hacer**
<bloque instrucciones>
fin_para

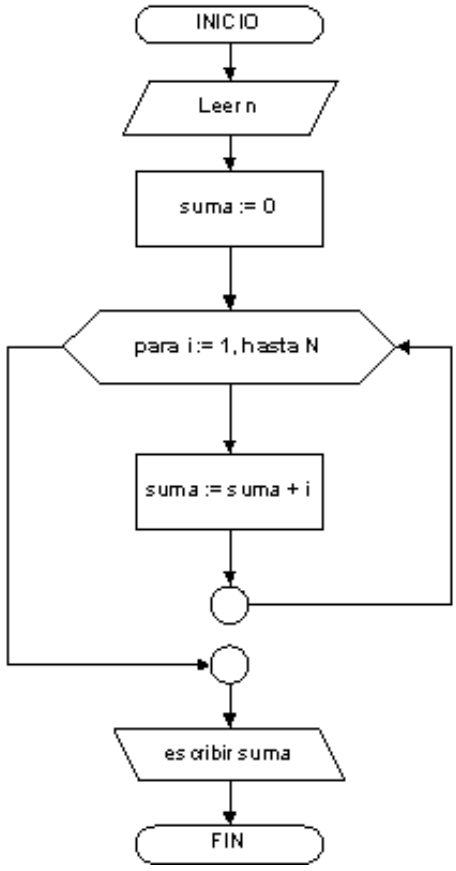
Donde **<variable>** es la *variable contadora* del ciclo, la cual debe ser de tipo entero. **<lim_inf>** es el valor inicial que toma la *variable contadora*. **<lim_sup>** es el último valor que toma la *variable contadora*; cuando el valor de la variable contadora supere este valor, el ciclo termina. **<bloque instrucciones>** es el conjunto de instrucciones que se ejecuta en cada iteración, mientras la *variable contadora* no sobrepase el **<lim_sup>**.

Ejemplos.

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

```
n: entero
suma: entero
i: entero
leer n

suma:= 0

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

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

Diagrama de flujo:



Ejemplo 2. Calcular las primeras tres filas de la tabla de multiplicar de un número dado.

ANALISIS DEL PROBLEMA:

Variables Conocidas	Un número.
Variables Desconocidas	Tres números.
Condiciones	Los números buscados son el resultado de multiplicar un número conocido, por los números entre uno y tres.

ESPECIFICACIÓN:

Entradas	n $\in$ Enteros (**n** es el número dado).
Salidas	a_1, a_2, a_3 $\in$ Enteros, ($\mathbf{a_i}$ es el i-esimo múltiplo del número dado).
Condiciones	$a_i = n * i$ para $1 <= i <= 3$

DISEÑO:

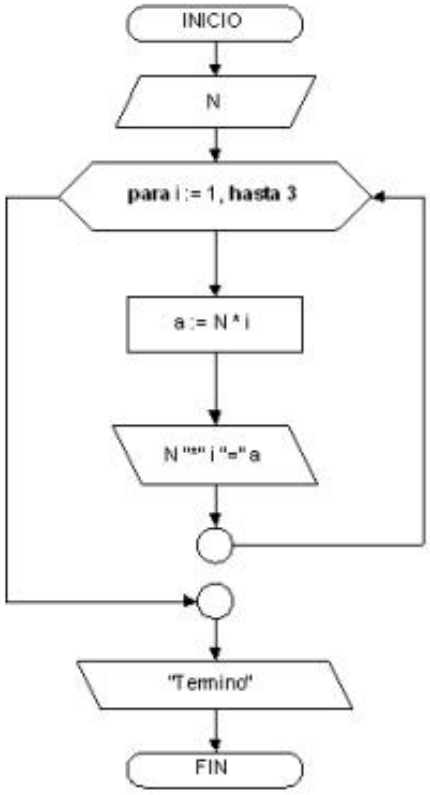
Primera División:

Inicio
Paso 1. Leer el número a calcularle la tabla de multiplicar
Paso 2. Para los números entre uno y tres calcular el múltiplo del número
Fin

División Final:

```
1  n: entero
2  a: entero
3  i: entero
4  escribir ( “Ingrese el número a calcularle la tabla de multiplicar:” )
5  leer (n)

6  para i :=1 hasta 3 hacer
7     a :=n * i
8     escribir (n, “*”, i, “=”, a, cambio_linea)
9  fin_para

10 escribir (“Termino...” )
```

Diagrama de flujo:



PRUEBA DE ESCRITORIO:

Este algoritmo cuenta con diez (10) líneas, las tres primeras, son para definir las variables usadas y las últimas son las instrucciones que son aplicadas sobre dichos datos. De esta manera la prueba de escritorio se debe realizar solamente sobre las líneas 4-10, teniendo en cuenta los valores para las variables.

LINEA	n	I	A	ENTRADA	SALIDA

4					Ingrese el número a calcularle la tabla de multiplicar:
5	3			3	
6		1			
Como es la primera vez que llega a esta línea, se asigna en la variable contadora el límite inferior. Ahora se comprueba si la variable contadora es menor o igual al límite superior. En este caso es cierto entonces se ejecuta el bloque de acciones del ciclo para, es decir, se pasa a la línea					
7			3		
8					3 * 1 = 3
9		2			
Se incrementa la variable contadora y se vuelve a la línea de inicio del ciclo para, es decir, línea 6.					
6	Se comprueba si la variable contadora es menor o igual al límite superior. En este caso es cierto entonces se ejecuta el bloque de acciones del ciclo para, es decir, se pasa a la línea 7.				
7			6		
8					3 * 2 = 6
9		3			
Se incrementa la variable contadora y se vuelve a la línea de inicio del ciclo para, es decir, línea 6.					
6	La variable contadora es menor que el límite superior se pasa a la línea 7				
7			9		
8					3 * 3 = 9
9		4			
Se incrementa la variable contadora y se vuelve a la línea de inicio del ciclo para, es decir, línea 6.					
6	La variable contadora no es menor que el límite superior se pasa a la línea siguiente al **fin_para**, es decir, a la línea 10.				
10					Termino...

5.5 TIPO DE VARIABLES ÚTILES PARA LA ITERACIÓN

Cuando se diseñan algoritmos que incluyen estructuras de control repetitivas, existen ciertas variables que cumplen una función específica en cada iteración del ciclo; las más comunes son:

- Las variables contadoras
- Las variables acumuladoras
- Las variables bandera.

5.5.1 Variables contadoras

Como su nombre lo indica estas variables se usan fundamentalmente para contar, por lo

tanto deben ser de tipo entero. Un ejemplo de este tipo de variables es la variable de control en un **ciclo para**.

Una *variable contadora* se incrementa (o decrementa) en un valor constante en cada iteración del ciclo. Es así como en los algoritmos presentados anteriormente para resolver el problema de calcular la suma de los números naturales desde 1 hasta **n**, la variable **i** es una *variable contadora*.

Ejemplo.
Desarrollar un algoritmo que imprima los números impares en orden descendente que hay entre 1 y 100.

Algoritmo Solución

```
i: entero
i :=99
mientras (i >= 1 hacer
   escribir ( i, ',' )
   i := i – 2
fin_mientras
```

En este caso **i** es una *variable contadora*, ya que en cada iteración del ciclo la variable es decrementada en una cantidad fija, 2 en este caso.

5.5.2 Variables acumuladoras

La función de una *variable acumuladora* es almacenar valores numéricos que generalmente se suman (o multiplican) en cada iteración, por lo tanto la variable debe ser de tipo entero o real. Por ejemplo, en los diferentes algoritmos presentados para solucionar el problema de calcular la suma de los números naturales desde 1 hasta **n**, la variable **suma** es una *variable acumuladora*.

Ejemplo.
Calcular las suma de los cuadrados de los números entre 1 y 100.

Algoritmo Solución

```
i: entero
suma: entero

i :=1
suma := 0
mientras (i <= 100)hacer
   suma := suma + i * i
   i := i + 1
fin_mientras

escribir ("La suma de los cuadrados hasta 100 es:", suma)
```

En este caso **suma** es una *variable acumuladora* mientras que la variable **i** es una *variable contadora*.

5.5.3 Variables bandera

Una *variable bandera* es utilizada dentro de la condición de un ciclo, para determinar cuándo un ciclo se sigue iterando o cuando no. De esta manera una *variable bandera* debe ser de tipo booleano.

Ejemplo.
Realizar un programa que lea una serie de números reales y los sume. El programa debe preguntar al usuario cuando desea ingresar un siguiente dato y si el usuario responde que no desea ingresar más datos el programa debe confirmar la respuesta. Si el usuario desea continuar ingresando datos se debe seguir solicitando datos y si el usuario confirma su deseo de salir, el programa debe mostrar la suma de los datos leídos y terminar.

ESPECIFICACIÓN:

$$suma = \sum_{i=1}^{n} datos_i$$

Donde, **datos** es la colección de **n** números reales que el usuario ingresa hasta que decide no continuar ingresando datos y **suma** es la sumatoria de dichos números y pertenece a los reales.

Algoritmo Solución

```
bandera: booleano
suma: real
dato: real
c: caracter

bandera := VERDADERO
suma :=0.0

mientras (bandera)hacer
   escribir ("Ingrese un dato:")
   leer (dato)
   suma := suma + dato
   escribir ("Desea continuar ingresando datos (S/N):")
   leer (c)
   si (c = 'N' | c = 'n') entonces
      bandera :=FALSO
   fin_si
fin_mientras

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

Vale la pena recordar que una variable de tipo booleano toma valores de verdad y por lo tanto, por sí sola es una expresión lógica. Adicionalmente, la expresión lógica **bandera = verdadero** es equivalente a la expresión lógica **bandera**.

5.6 CORRESPONDENCIA ENTRE CICLOS

En la teoría matemática de programación sólo es necesario un tipo de ciclo, en esta sección se explican las correspondencias que hacen posible esta afirmación, tomando como ciclo referencia el **ciclo mientras-hacer**.

5.6.1 Correspondencia entre el ciclo mientras y el ciclo haga-mientras

La diferencia fundamental entre los ciclos **mientras** y **haga-mientras**, es que en el segundo se ejecuta por lo menos una vez el <bloque de instrucciones>, mientras que en el primero hay la posibilidad de que no se ejecute alguna vez.

El ejecutar el bloque de acciones una vez antes del ciclo **mientras** permite modelar un ciclo **haga-mientras**, es decir:

haga <bloque> **mientras** (<condición>)	**mientras** <condición> **hacer** <bloque> **fin_mientras**

5.6.2 Correspondencia entre el ciclo mientras y el ciclo repita-hasta

Para determinar la correspondencia entre el ciclo **mientras** y el ciclo **repita-hasta**, primero se debe establecer la correspondencia entre los ciclos **haga-mientras** y **repita-hasta**.

La diferencia fundamental entre los ciclos **haga-mientras** y **repita-hasta** es que en el primero se repite la ejecución del <bloque de instrucciones> si la <condición> es evaluada como **verdadero** mientras que en el segundo se repite si es evaluada como falso.

De esta manera se tiene la siguiente correspondencia:

haga <bloque> **mientras** (<condición>)	**repita** <bloque> **hasta** <condición>

Por lo tanto se obtiene la correspondencia:

repita <bloque> **hasta**<condición>	**mientras** <condición> **hacer** <bloque> **fin_mientras**

5.6.3 Correspondencia entre el ciclo para y el ciclo mientras

Formalmente, un ciclo para es una forma abreviada de un ciclo **mientras**, precedido por una asignación y que en cada iteración incrementa una variable. Por lo tanto, el siguiente ciclo para:

```
para <variable> := <lim inf> hasta <lim sup> hacer
   <bloque>
fin _para
```

Es la abreviación del siguiente bloque de acciones:

```
<variable> := <lim inf>
mientras <variable> < <lim sup> hacer

   <bloque>
    <variable> := <variable> + 1
fin _mientras
```