ESTRUCTURAS ITERATIVAS

ESTRUCTURAS ITERATIVAS

OBJETIVOS

Aprender a resolver problemas mediante la ejecución repetida de una secuencia de proposiciones llamados bucle o estructuras repetitivas o iterativas.

Distinguir las diferentes estructuras de repetición utilizadas en problemas con bucles: mientras, repetir… mientras, para.

Analizar las diferencias entre cada una de las estructuras de repetición.

Analizar la correspondencia entre cada una de las estructuras de repetición.

INTRODUCCION

Las estructuras ITERATIVAS o de repetición, permiten la ejecución de **una instrucción** o una **secuencia de instrucciones** (<bloque de instrucciones>) tantas veces como sea necesario. 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 y cuándo no. A cada ejecución del bloque de instrucciones se le conoce como una **iteración**.

1. TIPOS DE ITERACIÓN

Existen tres tipos principales de sentencias de repetición:

- **Bucle MIENTRAS…**
- **Bucle REPETIR… MIENTRAS**
- **Bucle PARA**

A continuación se describe cada una de ellas.

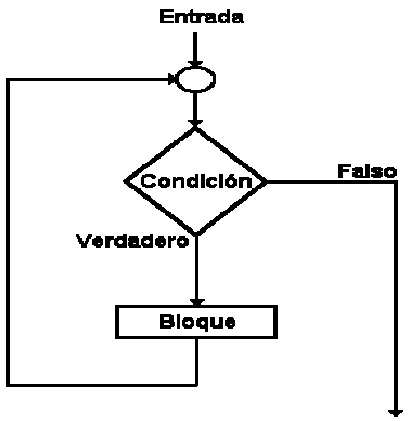
1.1 BUCLE MIENTRAS

El **bucle mientras** permite ejecutar 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 **SÍ** se cumple, el bloque **SÍ** 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 se 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 como **verdadera**, la instrucción se ejecutará indefinidamente, es decir, un número infinito de veces.

La forma general del bucle 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 uno hasta el número dado.

ESPECIFICACIÓN:

Entradas	n $\in$ Enteros, n $\geq$ 0 (n es el número dado).
Salidas	suma $\in$ Enteros, suma $\geq$ 0 suma es la sumatoria de los primeros n números naturales. $suma = \sum_{i=1}^{n} i$

DISEÑO:

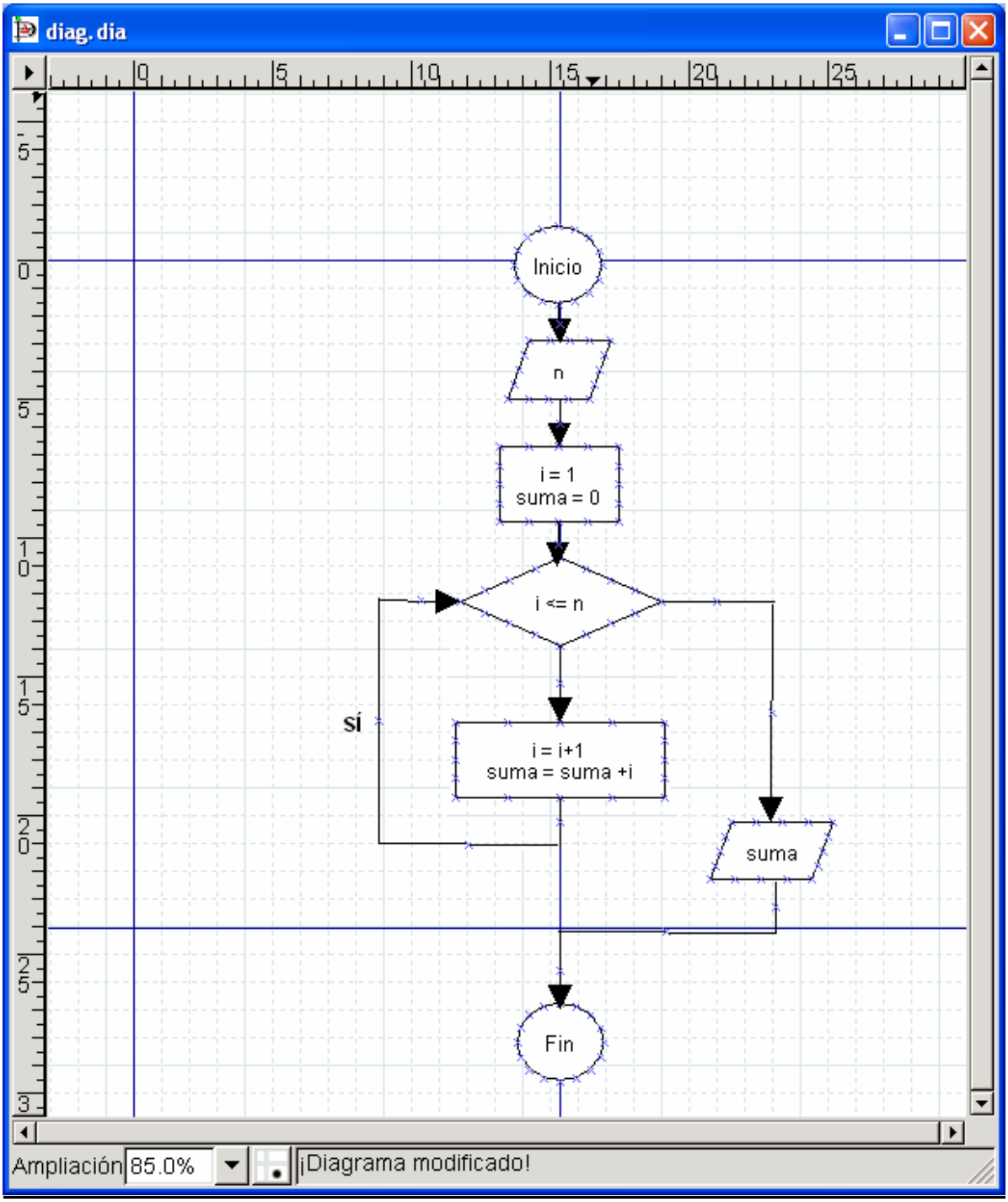
Primera Aproximación:

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

Refinamiento:

```
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 ( "Escriba el numero: " )
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 1 */


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


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

Diagrama de Flujo:



ANÁLISIS DEL ALGORITMO:

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. Veamos las instrucciones entre las líneas 4 y 12, suponiendo que el usuario introduzca el valor 5:

```
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 ( "Escriba el numero: " )
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 1 */

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

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

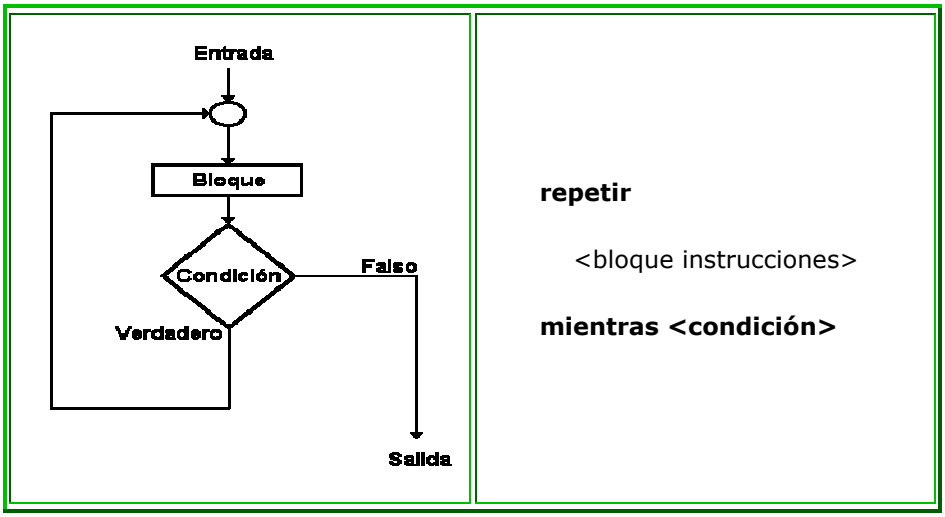
LINEA	n	i	suma	ENTRADA	SALIDA
4					Escriba el numero
5	5			5	
6			0		
7		1			
8	La condición es evaluada a verdadero, por lo tanto se ejecuta el bloque de acciones del bucle, es decir, pasa a la línea 9.				
9			1		
10		2			
11	Se salta hasta la línea que contiene la condición del bucle mientras en ejecución, es decir, hasta la línea 8				
8	La condición es evaluada a verdadero, por lo tanto se ejecuta el bloque de acciones del bucle, es decir, pasa a la línea 9.				
9			3		
10		3			
11	Se salta hasta la línea que contiene la condición del bucle 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 bucle, es decir, pasa a la línea 9.				
9			6		
10		4			
11	Se salta hasta la línea que contiene la condición del bucle 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 bucle, es decir, pasa a la línea 9.				
9			10		
10		5			
11	Se salta hasta la línea que contiene la condición del bucle 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 bucle, es decir, pasa a la línea 9.				
9			15		
10		6			
11	Se salta hasta la línea que contiene la condición del bucle 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 bucle y este termina, es decir, pasa a la línea 12, la línea siguiente a la línea del **fin_mientras** del bucle.				
12					La suma es: 15

1.2 BUCLE REPETIR... MIENTRAS

El **bucle REPETIR... MIENTRAS** es similar al **bucle mientras**, la diferencia radica en el momento de evaluación de la condición.

En el **bucle REPETIR... 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 como verdadero, y no se ejecuta más si se evalúa como falso.

La forma general del bucle **REPETIR... MIENTRAS** es la siguiente:



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

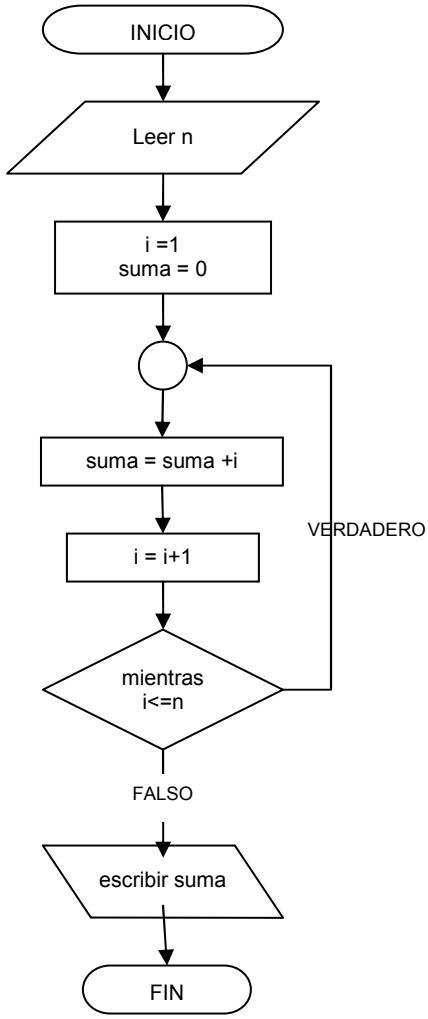
Ejemplos

Ejemplo 1.

El problema de calcular la suma de los números naturales desde 1 hasta n (enunciado anteriormente), se puede solucionar usando el **bucle REPETIR... 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 ( "Introduzca  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 1 */

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:



ANÁLISIS DEL ALGORITMO:

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		1			
8	Se ejecuta la instrucción **haga**, es decir pasa a la línea 9				
9			1		
10		2			
11	La condición es evaluada a verdadero, por lo tanto se salta hasta la línea que contiene la condición del bucle **haga_mientras** en ejecución, es decir, hasta la línea 8				
8	Se ejecuta la instrucción **haga**, es decir pasa a la línea 9				
9			3		
10		3			
11	La condición es evaluada a verdadero, por lo tanto se salta hasta la línea que contiene la condición del bucle **haga_mientras** en ejecución, es decir, hasta la línea 8				
8	Se ejecuta la instrucción **haga**, es decir pasa a la línea 9				
9			6		
10		4			
11	La condición es evaluada a verdadero, por lo tanto se salta hasta la línea que contiene la condición del bucle **haga_mientras** en ejecución, es decir, hasta la línea 8				
8	Se ejecuta la instrucción **haga**, es decir pasa a la línea 9				
9			10		
10		5			
11	La condición es evaluada a verdadero, por lo tanto se salta hasta la línea que contiene la condición del bucle **haga_mientras** en ejecución, es decir, hasta la línea 8				
8	Se ejecuta la instrucción **haga**, es decir pasa a la línea 9				
9			15		
10		6			
11	La condición es evaluada a falso, por lo tanto este bucle termina y se salta a la línea 12				
12					La suma es: 15

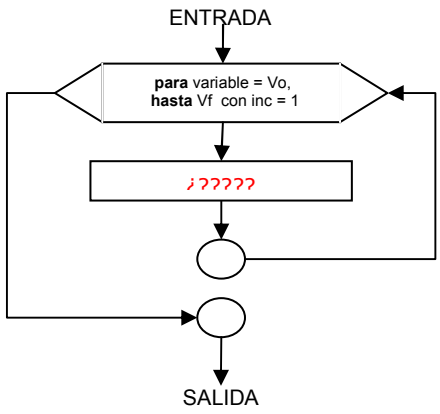
1.3 BUCLE PARA

El **bucle para** ejecuta un bloque de instrucciones un **número determinado** de veces. Este número de veces está determinado por una *variable controladora* que toma valores desde un *valor inicial* hasta un *valor final*.

En cada bucle después de ejecutar el bloque de instrucciones, la *variable controladora* es incrementada en un valor llamado *incremento* automáticamente y en el momento en que la variable sobrepasa el *límite superior* el bucle termina.

De la definición de **bucle 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 bucle para es la siguiente:



para (<variable> = <Voo> **hasta** <Vf> **con incremento** = inc =1)
hacer
<bloque instrucciones>
fin_para

Donde:

<variable> es la *variable controladora* del bucle,
<Vo> es el valor inicial que toma la *variable controladora*.
<Vf> es el último valor que toma la *variable controladora*; cuando el valor de la variable contadora supere este valor, el bucle termina.
<bloque instrucciones> es el conjunto de instrucciones que se ejecuta en cada iteración, mientras la *variable controldora* no sobrepase el **<lim_sup>**.

Casos:

- Cuando <**Vo**> es menor que <**Vf**> ocurre lo siguiente (si <inc> = 1):

  1. La variable contadora se inicializa con <Vo>
  2. Se ejecuta <**bloque de instrucciones**>
  3. Se incrementa automáticamente en 1 la variable contadora del bucle.
  4. Si el valor de contador del bucle es menor o igual que <Vf> se vuelve de nuevo al paso 2. De otro modo se abandona el bucle.

- Es importante observar que el valor final de la variable contadora supera a <Vf> para que pueda finalizar el bucle

- Cuando <**Vo**> es mayor que <**Vf**> el bucle termina sin ejecutarse nunca el <**bloque de instrucciones**>., excepto si el incremento <inc> es negativo. Tenga en cuenta que no se genera error al correr el programa

  Ejemplo:

  para (x=5 hasta 4 con inc = 1)
  hacer

  Esta línea de código nunca se ejecuta.

- Pero:

  para (x=5 hasta 4 con inc = -1)
  hacer

ejecuta el bucle perfectamente.

- Tanto <**Vo**> como <**Vf**> pueden ser expresiones como en el siguiente **ejemplo:**

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

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

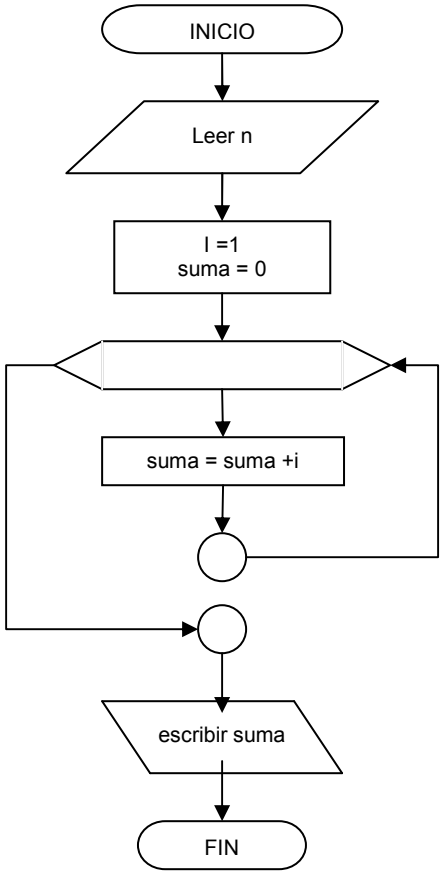
Ejemplos.

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

```
 1 n: entero  /* se define la variable para un número entero*/
 2 suma: entero /* se define la variable para la suma*/
 3 i: entero /* se define la variable la variable contadora */
 4 escribir("ingrese el número:")
 5 leer n /* lee el primer número */

 6 suma = 0

 7 para(i  =1 hasta n)hacer
 8    suma  =suma + i
 9 fin_para

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

Nótese que se requieren menos instrucciones que en las anteriores estructuras dado que el incremento de **i** se hace automáticamente en la instrucción 7 al repetir el bucle.

Diagrama de flujo:



Análisis del Algoritmo:

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

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

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-ésimo múltiplo del número dado).
Condiciones	$a_i = n * i$ para $1 <= i <= 3$

DISEÑO:

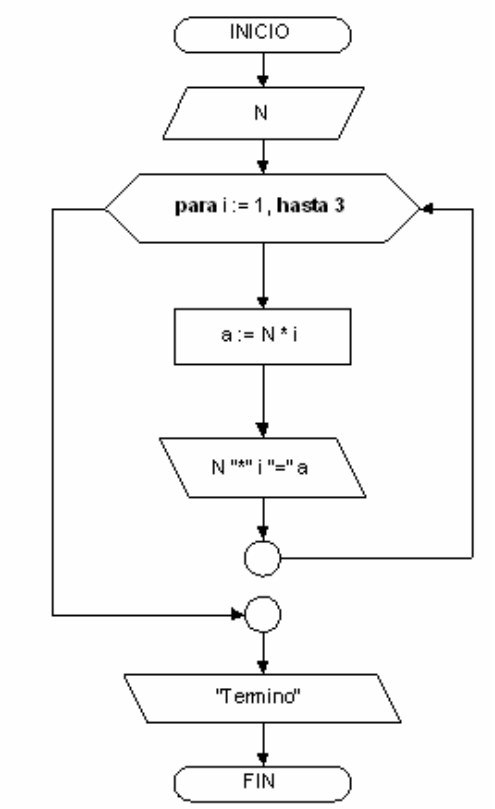
Primera Aproximación:

```
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
```

Refinamiento:

```
 1  n: entero
 2  a: entero
 3  i: entero
 4  escribir ( "Introduzca el número para calcular su 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:



ANALISIS DEL ALGORITMO:

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 bucle para, es decir, se pasa a la línea					
7			3		
8					3 * 1 = 3
Se incrementa la variable contadora y se vuelve a la línea de inicio del bucle 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 bucle para, es decir, se pasa a la línea 7.	2			
7			6		
8					3 * 2 = 6

LINEA	N	I	A	ENTRADA	SALIDA
Se incrementa la variable contadora y se vuelve a la línea de inicio del bucle para, es decir, línea 6.					
6	La variable contadora es menor que el límite superior se pasa a la línea 7	3			
7			9		
8					3 * 3 = 9
Se incrementa la variable contadora y se vuelve a la línea de inicio del bucle 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.	4			
10					Termino...

2.

3. 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 bucle, las más comunes son:

- Las variables contadoras
- Las variables acumuladoras
- Las variables bandera (flag, switch)

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 **bucle para**.

Una *variable contadora* se incrementa (o decrementa) en un valor constante en cada iteración del bucle. 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 repetición del bucle la variable es decrementada en una cantidad fija, 2 en este caso.

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*.

VARIABLES BANDERA (FLAG, SWITCH)

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

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: entero
dato: real
c: caracter

bandera = 1
suma = 0.0

mientras (bandera = 1) hacer
   escribir ("Introduzca un dato:")
   leer (dato)
   suma = suma + dato
   escribir ("Desea continuar ingresando datos (S/N):")
   leer (c)
   si (c = 'N' OR c = 'n') entonces
      bandera = 0
   fin_si
fin_mientras

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

4.
5. CORRESPONDENCIA ENTRE BUCLES

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

Correspondencia entre el bucle mientras y el bucle repetir -mientras

La diferencia fundamental entre los bucles **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 bucle **mientras** permite modelar un bucle **haga-mientras**, es decir:

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

Correspondencia entre el bucle para y el bucle mientras

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

```
para <variable> = <Vo> hasta <Vf> hacer
   <bloque>
fin _para
```

Es la abreviación del siguiente bloque de acciones:

```
<variable> = <Vo>
mientras <variable> <= <Vf> hacer
   <bloque>
    <variable> = <variable> +  1
fin _mientras
```

Cuando usar estructuras de control definido o indefinidos

El **bucle para** se conoce comúnmente como **estructura de control definida,** ya que los valores iniciales y finales especificados para la variable contadora que controla el bucle determina de manera exacta el número de veces que se ejecuta el bucle.

Para utilizar un **bucle para** al resolver un algoritmo se debe determinar el número exacto de veces que se va a ejecutar el bucle. En el ejemplo de calcular la suma de los números de 1 hasta n. Se sabe que el bucle se repetirá n veces:

n = ((Vf - Vo)\inc) + 1

Resumen

Las estructuras de control cíclico permiten controlar la ejecución repetida de una secuencia de instrucciones.

El **bucle mientras** permite ejecutar un bloque de instrucciones mientras que la evaluación de una expresión lógica de cómo resultado verdadero.

El **bucle repetir-mientras** permite ejecutar un bloque de instrucciones por lo menos una vez, después evalúa la condición para ejecutar de nuevo el bucle si la condición es verdadera.

El **bucle para** ejecuta un bloque de instrucciones un número determinado de veces.

EJERCICIOS

1. Imprimir un listado con los números del 1 al 100 cada uno con su respectivo cuadrado
2. Imprimir un listado con los números impares desde 1 hasta 999 y seguidamente otro listado con los números pares desde 2 hasta 1000
3. Imprimir los números pares desde N (siendo N un número par que se lee) en forma descendente hasta 2.
4. Imprimir los 100 primeros números de Fibonacci. Recuerde que un número de Fibonacci se calcula como la suma de los dos anteriores así:
   0, 1, 1, 2, 3, 5, 8,13…
5. Imprimir los números de 1 a N (siendo N un número que se lee) cada uno con su respectivo factorial.
6. Calcular el factorial de un número N (siendo N un número que se lee).
7. Calcular el factorial de 10 números diferentes cuyos valores se leen.
8. Leer 20 números y encontrar el mayor y el menor valor leídos.
9. Leer un dato y almacenarlo en la variable **n**. Calcular el valor de 2 elevado a la potencia **n**
10. Leer un dato y almacenarlo en la variable **n**, leer otro dato y almacenarlo en la variable **x**. Calcular el valor de **x** elevado a la potencia **n**.
11. Una papelería debe imprimir una lista de los valores para diferentes cantidades de fotocopias a sacar. El precio unitario de cada fotocopia debe leerse. Imprimir un listado teniendo en cuenta que se tiene una política de descuento para cantidades que se

obtengan del mismo original así: el 12% para fotocopias entre 100 y 200, del 15% para fotocopias entre 201 y 400, y del 18% para fotocopias por cantidades mayores a 400.

12. En 1994 el país A tiene una población de 25 millones de habitantes y el país B de 19.9 millones. Las tasas de crecimiento de la población son de 2% y 3% respectivamente. Desarrollar un algoritmo para informar en que año la población del país B supera a la de A.

BIBLIOGRAFÍA

1. **Becerra César**. Algoritmos, Editorial César A. Becerra, 1993.

2. **Goldstein Larry Joel.** Turbo Pascal. Editorial Prentice-Hall Hispanoamericana, 1993.

3. **Konvalina John y Stantley Willeman**. Programación con Pascal. Editorial McGraw-Hill, 1989.

TRADUCCIÓN DE LAS ESTRUCTURAS DE REPETICION

	SEUDOCODIGO	C++
Bucle mientras	**mientras** <condición> **hacer** <bloque instrucciones> **fin_mientras**	**while(**<condición>**)** **{** <bloque instrucciones> **}**
Bucle haga-mientras	**haga** <bloque instrucciones> **mientras <condición>**	**do** **{** <bloque instrucciones> **}** **while(**<condición>**);**
Bucle para	**para (i** := <lim_inf> **hasta** <lim_sup>**) hacer** <bloque instrucciones> **fin_para**	**for (i =** <lim_inf>**; i <=** <lim_sup>**; i ++)** **{** <bloque instrucciones> **}**