UNIVERSIDAD DISTRITAL
FRANCISCO JOSE DE CALDAS

```
static void Main(string[] args)
{
    int k;

    k = 1;

    while (k <= 10)
    {
        Console.WriteLine(k);

        k = k + 1;
    }

}
```

inicialización

Condicion de terminacion (ciclo mientras que)

incremento

Estructuras de Repetición

2013
Transversal de Programación Básica
Proyecto Curricular de Ingeniería de Sistemas

Objetivos

- Aprender a construir grandes y complejos problemas a través de la ejecución repetida de una secuencia de proposiciones llamados ciclos o estructuras repetitivas.
- Distinguir las diferentes estructuras de repetición utilizadas en problemas con ciclos: mientras, haga-mientras, repita-hasta, para.
- Analizar las diferencias entre cada una de las estructuras de repetición.
- Analizar la correspondencia entre cada una de las estructuras de repetición.

Introducción

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 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:

- **Ciclo mientras**
- **Ciclo haga-mientras**
- **Ciclo para**

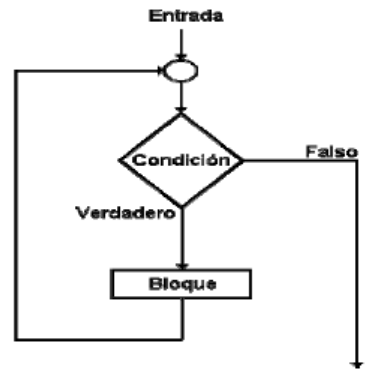
A continuación se describe cada una de ellas.

1.1 CICLO MIENTRAS

El **ciclo 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 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.

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

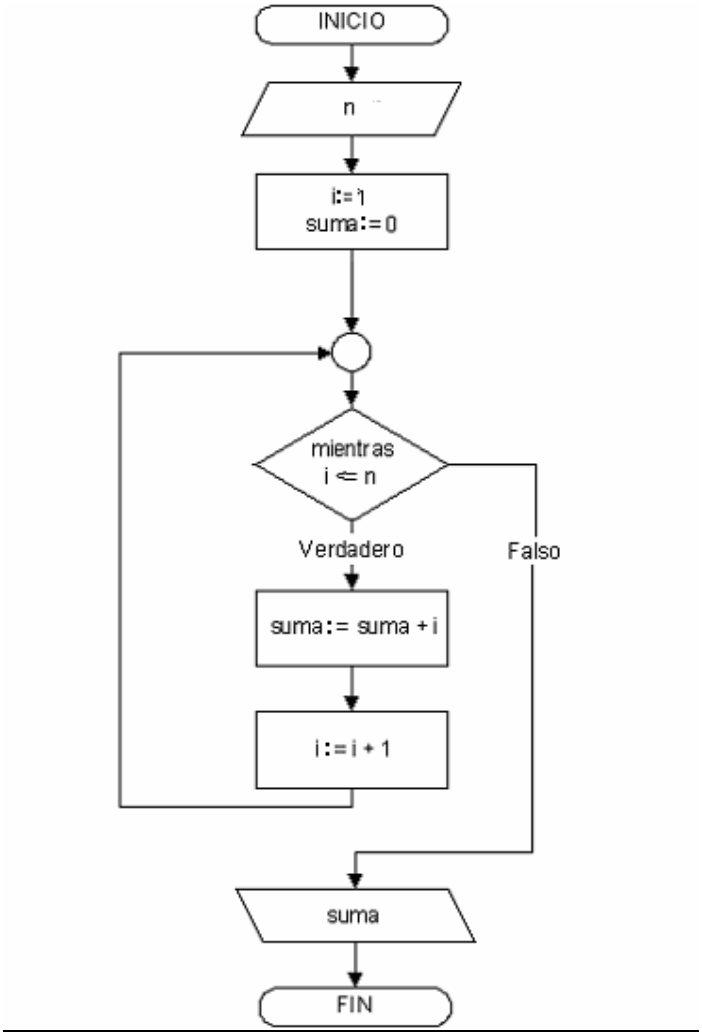
Diseño:
Primera División:

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 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:



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		1			
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 cido 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 cido 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 cido 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 cido 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 cido 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

Análisis 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 ∈Enteros, (a, b son los números dados).
Salidas	mcd ∈ 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

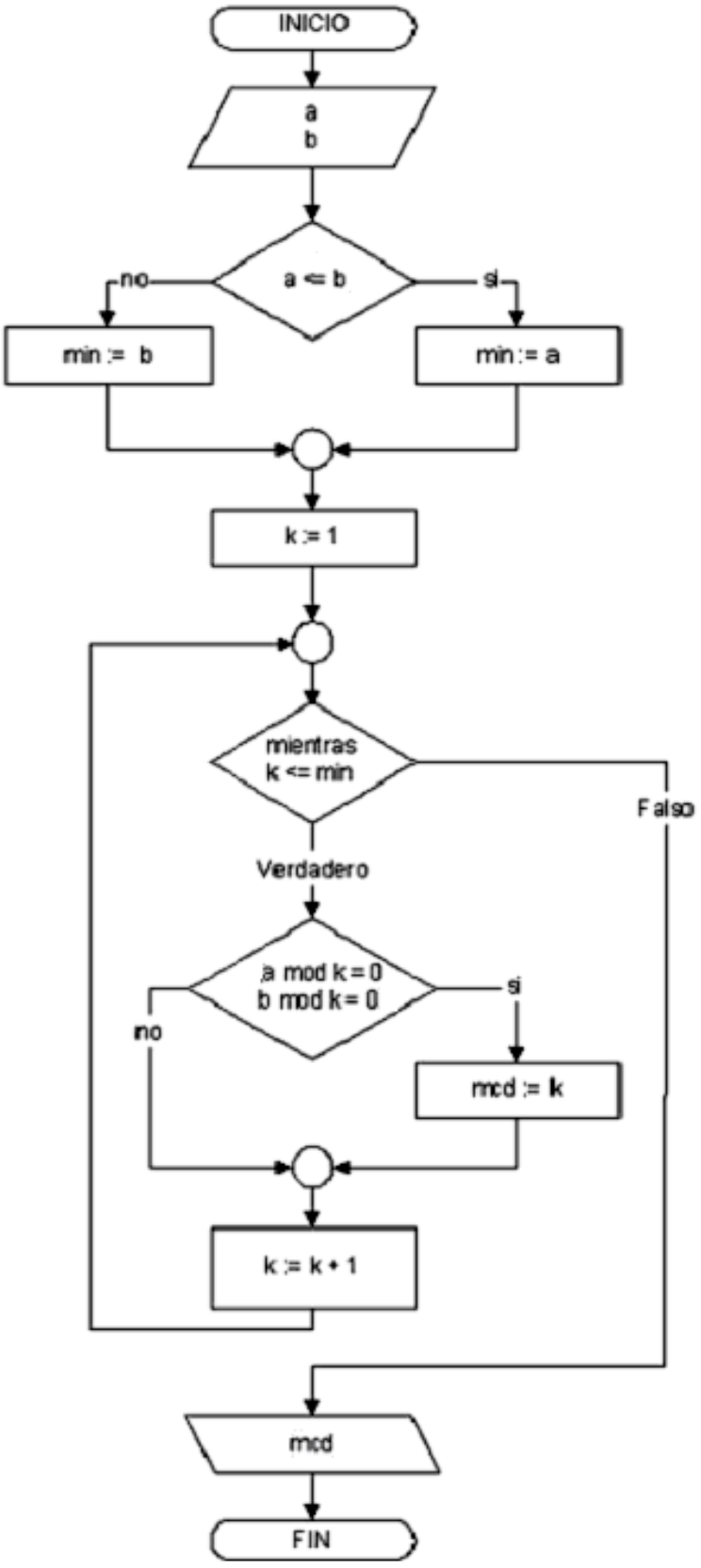
Segunda División:

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:

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

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

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

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

16 mientras (k <= min)hacer
17    si (a mod k = 0) & (b mod k = 0) entonces
18       mcd := k
19    fin_si
20     k :=  k + 1 /*incrementa k en 1 para tomar el siguiente número en la próxima iteración */
21 fin_mientras

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

Diagrama de Flujo:


INICIO
a
b
a <= b
no
si
min := b
min := a
k := 1
mientras
k <= min
Falso
Verdadero
a mod k = 0
b mod k = 0
si
no
mcd := k
k := k + 1
mcd
FIN

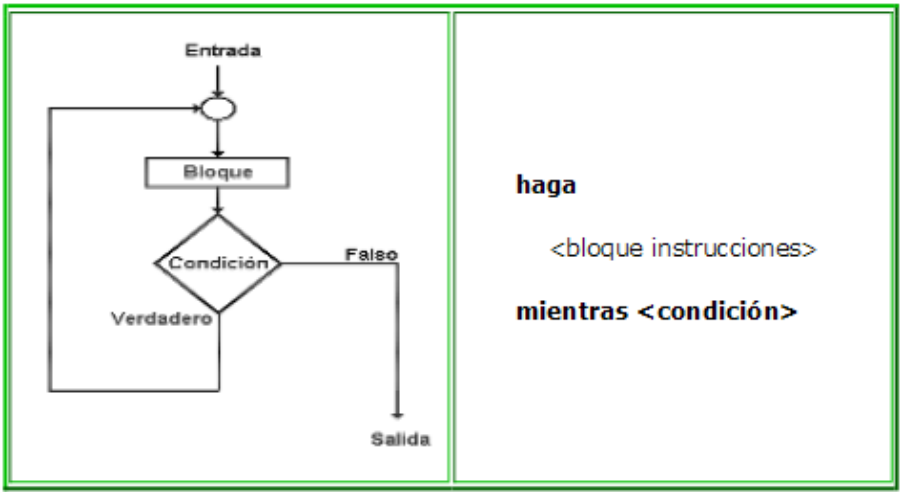
Prueba De Escritorio:

Este algoritmo cuenta con veintidós (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.						
13			6				
14	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.						
20				2			
21	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.						
20				3			
21	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.						
20				4			
21	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.						
19	Se salta a la siguiente línea al **fin_si**, línea 21.						
20				5			
21	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.						
19	Se salta a la siguiente línea al **fin_si**, línea 21.						
20				6			
21	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.						
19	Se salta a la siguiente línea al **fin_si**, línea 21.						
20				7			
21	Se retorna a la línea de inicio del ciclo **mientras**, línea 16						
16	La condición evalúa a falso (7<=6), luego no se ejecuta el bloque de acciones del ciclo **mientras** y se salta a la línea siguiente al **fin_mientras**, línea 22						
22							El máximo común divisor es: 2

1.2 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:



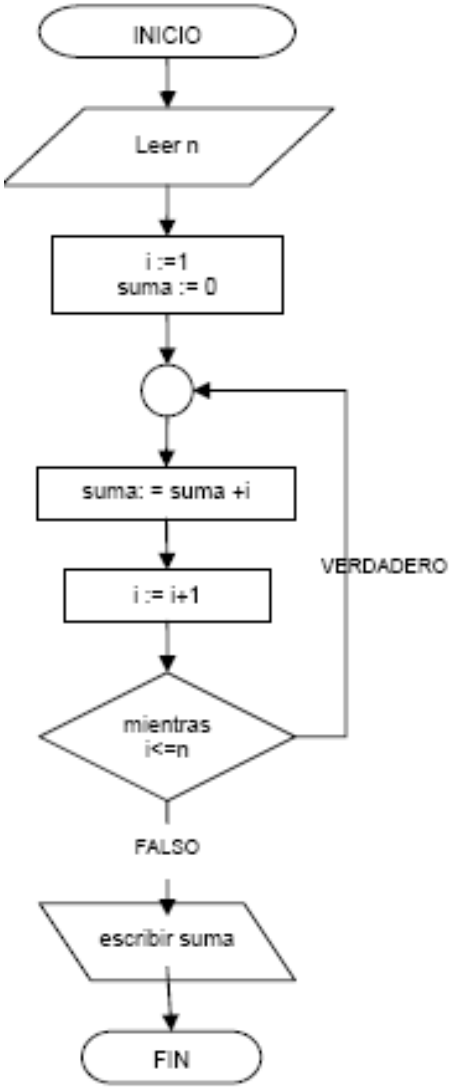
Donde, **<bloque instrucciones>** es el conjunto de instrucciones que se ejecuta y <condición> es la expresión lógica que determina si el bloque se ejecuta. Si la **<condición>** se evalúa como **verdadero** el bloque es ejecutado de nuevo y si es evaluada como **falso** no es ejecutado. Después de ejecutar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



Prueba de escritorio:

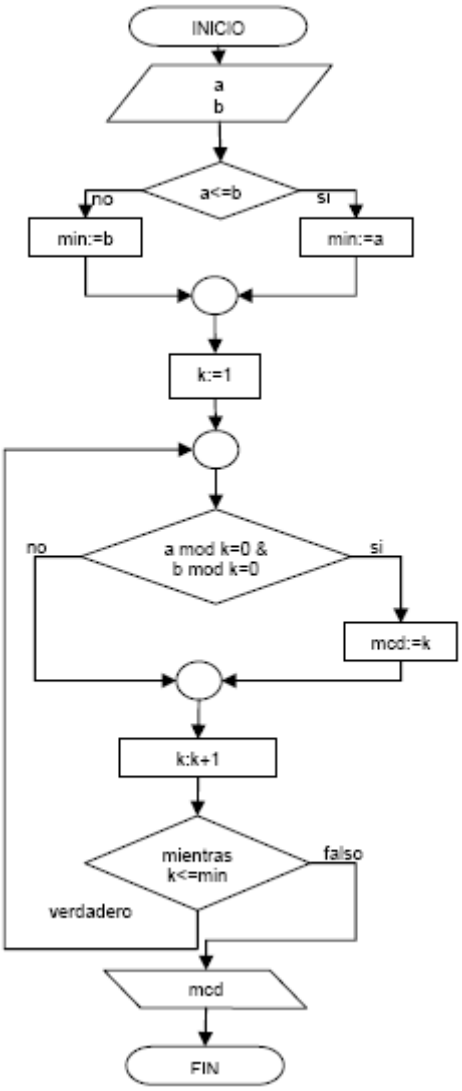
Este algoritmo cuenta con doce (12) líneas, las tres primeras, son para definir las variables usadas y las últimas nueve son las instrucciones que son aplicadas sobre dichos datos. De esta manera la prueba de escritorio se debe realizar solamente sobre las líneas 4-12, teniendo en cuenta los valores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

LINEA	n	i	suma	ENTRADA	SALIDA
4					Ingrese el número:
5	5			5	
6			0		
7		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 ciclo **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 ciclo **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 ciclo **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 ciclo **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 ciclo termina y se salta a la línea 12				
12					La suma es: 15

Ejemplo 2. El problema de calcular el máximo común divisor de dos números naturales, distintos de cero, se puede solucionar usando el **ciclo haga-mientras**. A continuación se describe el algoritmo solución:

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

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

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

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

16 haga
17    si (a mod k = 0) & (b mod k = 0) entonces
18       mcd := k
19    fin_si
20     k :=  k + 1 /*incrementa k en 1 para tomar el siguiente número en la próxima iteración */
21 mientras (k <= min)

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

Diagrama de flujo:



Prueba de escritorio:

Este algoritmo cuenta con veintidós (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

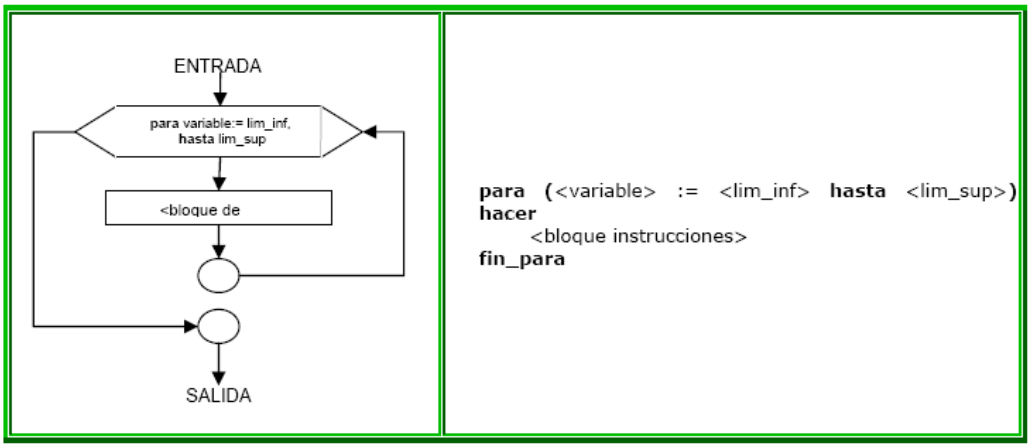
LINEA	a	b	Min	K	Mcd	ENTRADA	SALIDA
6							Ingrese el primer número:
7	8					8	
8							Ingrese el segundo número:
9		6				6	
10	La condición evalúa a falso, se pasa a la línea siguiente al sino, es decir, a la línea 13.						
13			6				
14	Se salta a la línea siguiente al fin_si, es decir, a la línea 15						
15				1			
16	Se ejecuta la condición haga						
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.						
20				2			
21	La condición evalúa a verdadero (2<=6), luego retorna a la línea de inicio del ciclo haga-mientras, línea 16						
16	Se ejecuta la condición haga						
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.						
20				3			
21	La condición evalúa a verdadero (3<=6), luego retorna a la línea de inicio del ciclo haga-mientras, línea 16						
16	Se ejecuta la condición haga						
17	La condición evalúa a falso, se ejecuta la línea siguiente al sino, línea 20.						
20				4			
21	La condición evalúa a verdadero (4<=6), luego retorna a la línea de inicio del ciclo haga-mientras, línea 16						
16	Se ejecuta la condición haga						
17	La condición evalúa a falso, se ejecuta la línea siguiente al sino, línea 20.						
20				5			
21	La condición evalúa a verdadero (5<=6), luego retorna a la línea de inicio del ciclo haga-mientras, línea 16						
16	Se ejecuta la condición haga						
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.						
20				6			
21	La condición evalúa a verdadero (6<=6), luego retorna a la línea de inicio del ciclo haga-mientras, línea 16						
16	Se ejecuta la condición haga						
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.						
20				7			
21	La condición evalúa a verdadero (7<=6), luego no retorna a la línea de inico del ciclo haga-mientras y se salta a la línea siguiente al fin_haga-mientras, línea 22						
22							El máximo común divisor es: 2

1.3 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:



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>**.
Casos:

- Cuando <**lim_inf**> es menor que <**lim_sup**> ocurre lo siguiente:

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

- Es de anotar que el valor final de la variable contadora queda incrementada por encima del <**lim_sup**> para que pueda finalizar el ciclo
- Cuando <**lim_inf**> es mayor que <**lim_sup**> el ciclo termina sin ejecutarse nunca el <**bloque de instrucciones**>. Tenga en cuenta que no se genera error al correr el programa

Ejemplo:

para (x:=5 hasta 4) **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 <**lim_inf**> y <**lim_sup**> respectivamente.

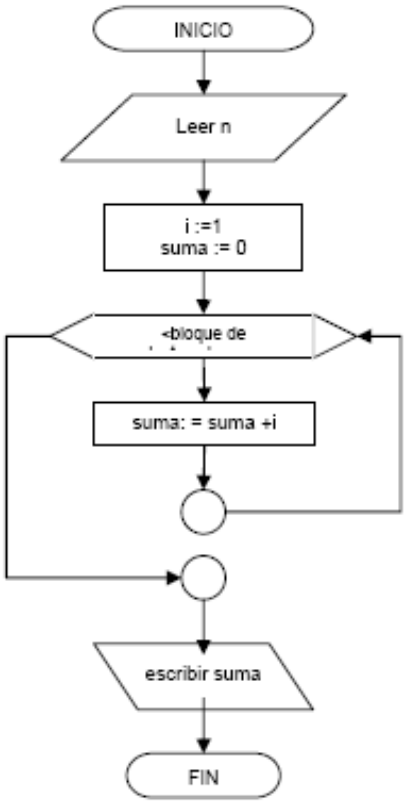
Ejemplos.

Ejemplo 1. El problema de calcular la suma de los números naturales desde 1 hasta n (enunciado anteriormente), se puede solucionar usando el **ciclo 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 ciclo.

Diagrama de flujo:



Prueba de escritorio

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 ciclo **para**, es decir, se pasa a la línea 8.					
7		1			
8			1		
9	Se vuelve a la línea de inicio del ciclo **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 ciclo **para**, es decir, se pasa a la línea 8.					
8			3		
9	Se vuelve a la línea de inicio del ciclo **para**, es decir, línea 7.				
7		3			
Se incrementa la variable contadora y se vuelve a la línea de inicio del ciclo para, es decir, línea 7. Como la variable contadora es menor que el límite superior se pasa a la línea 8					
8			6		
9	Se vuelve a la línea de inicio del ciclo **para**, es decir, línea 7.				
7		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					La suma es : 3

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

Análisis 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 ∈ Enteros (**n** es el número dado).
Salidas	$a_1, a_2, a_3 \in$ Enteros, (a_i es el i-ésimo múltiplo del número dado).
Condiciones	$a_i = n * i$ para $1 <= i <= 3$

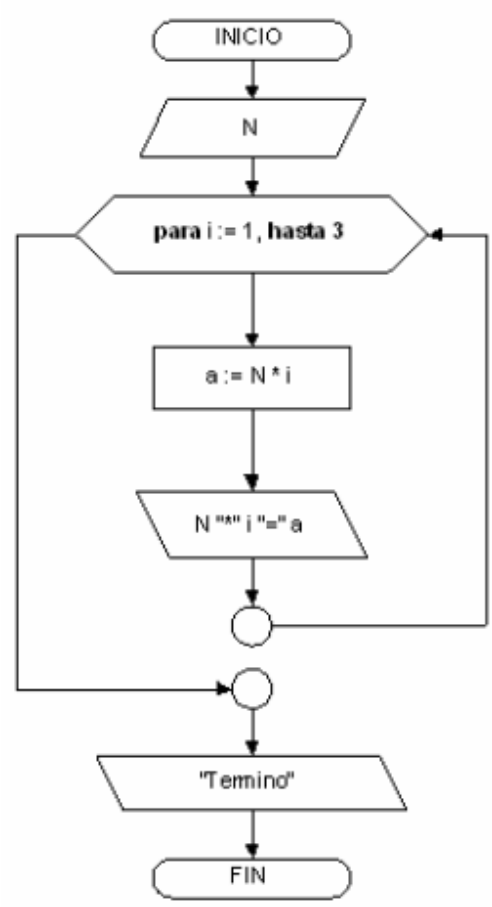
Diseño:
Primera División:

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
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.	2			
7			6		
8					3 * 2 = 6
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	3			
7			9		
8					3 * 3 = 9
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.	4			
10					Termino...

2. 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.

2.1 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.

2.2 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*.

2.3 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 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: 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 verdadero y falso. Note que la instrucción ***mientras*** *(bandera)* ***hacer*** es equivalente a ***mientras*** *(bandera = verdadero)* ***hacer.***

3. 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**.

3.1 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:

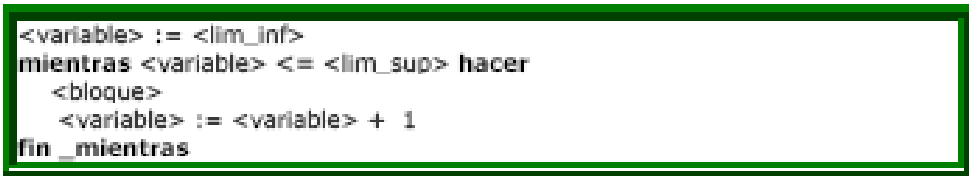
```
haga                                  mientras <condición> hacer
  <bloque>                              <bloque>
mientras (<condición>)                fin_mientras
```

3.2 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
```

3.3 CUANDO USAR ESTRUCTURAS DE CONTROL DEFINIDO O INDEFINIDOS

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

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

En el ejemplo de calcular el máximo común divisor de dos número naturales común divisor de dos números naturales no puede resolverse con un **ciclo para,** porque no se sabe de antemano cuantas veces se repetirá el ciclo.

Resumen

Las estructuras de control cíclico permiten controlar la ejecución repetida de una secuencia de instrucciones. El **ciclo mientras** permite ejecutar un bloque de instrucciones mientras que la evaluación de una expresión lógica de cómo resultado verdadero.

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

Lectura de Profundización

- Texto tomado de http://dis.unal.edu.co/~programacion/book/modulo2b.pdf
- Profundizar en http://eisc.univalle.edu.co/cursos/web/material/750001M/6/clase_6.pdf

Imágenes:

Las imágenes utilizadas en este documento fueron tomadas de www.google.com

Ejercicios de Aplicación

- http://users.dsic.upv.es/~onaindia/TEACHING/FUNDAMENTOS_FI/Ejercicios_Repeticion.pdf
- http://rsta.pucmm.edu.do/tutoriales/computacion/isc_201/leccion%2011.htm#ejercicios

Fuentes:

- https://sites.google.com/site/algoritmicai/estructuras-de-repeticion
- http://webdelprofesor.ula.ve/ingenieria/amoret/pd1/clase9.pdf
- http://informatica.uv.es/iiguia/FP/EstructurasRepeticion05.pdf
- http://rsta.pucmm.edu.do/tutoriales/computacion/isc_201/leccion%2011.htm
- http://es.scribd.com/doc/42704214/Estructura-de-repeticion-while
- http://msanchez.usach.cl/lcc/computacion/estructura-repeticion.pdf