ITERACIONES

Cuando se necesita repetir un proceso un numero determinado de veces es posible utilizar la instrucción *for* (PARA).

Ejemplo: Que hará el siguiente algoritmo?

```
INICIO
    Para i Desde 1 Hasta 5 Incrementando de a 1
        Escribir "Hola"
    Fin Para
FIN
```

Lo que hace es escribir 5 veces el texto "Hola", de una manera más elegante que haber puesto:

```
INICIO
    Escribir "Hola"
    Escribir "Hola"
    Escribir "Hola"
    Escribir "Hola"
    Escribir "Hola"
FIN
```

Que tal que en vez de 5 se quisiera repetir el mensaje 1000 veces?, usando el PARA sólo se necesitaria un simple cambio así:

```
INICIO
    Para i Desde 1 Hasta 1000 Incrementando de a 1
        Escribir "Hola"
    Fin Para
FIN
```

Un aspecto fundamental de esta instrucción es que la variable usada como indicador de las iteraciones (las repeticiones) va cambiando de valor en cada iteración y por tanto puede ser usada dentro del mismo algoritmo. Así por ejemplo, un algoritmo para escribir los números enteros del 1 al 10 puede ser:

```
INICIO
    Para i Desde 1 Hasta 10 Incrementando de a 1
        Escribir i
    Fin Para
FIN
```

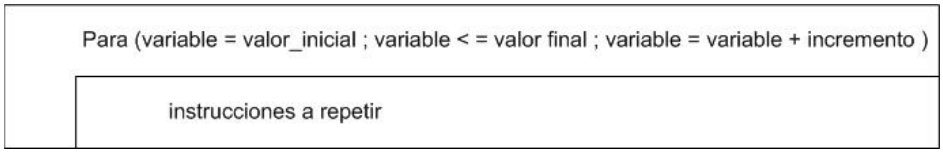
Como podría cambiarse este algoritmo para que en vez de los números del 1 al 10 mostrara los múltiplos de 3 hasta el 30? Una forma de hacerlo es la siguiente:

```
INICIO
        Para i Desde 3 Hasta 30 Incrementando de a 3
                Escribir i
        Fin Para
FIN
```

En general, la fórmula del PARA se escribe en pseudocódigo de la siguiente manera:

Para variable *Desde* valor_inicial *Hasta* valor_final *Incrementando de a* incremento

Y en diagrama de caja:



Que traducido a C queda:

for(variable = valor_inicial; variable <= valor_final; variable = variable + incremento)

A partir de esta fórmula es fácil intuir el número exacto de veces que se repetirá el proceso de la siguiente manera:

Número_de_repeticiones = (valor_final – valor_inicial) / incremento + 1

En caso que de un valor real (no entero) se redondea quitando los decimales.

Ejemplo: realizarle la prueba de escritorio al siguiente algoritmo

```
INICIO
        Para i Desde 8 Hasta 30 Incrementando de a 7
                Escribir i*2+1
        Fin Para
FIN
```

La variable utilizada en el PARA no necesariamente tiene que moverse aumentando de valor. Aunque suene extraño el incremento puede ser negativo, caso en el cual el valor inicial es mayor que el final. Así por ejemplo, para escribir los números de 80 a 50 de manera descendente se puede utilizar el siguiente algoritmo.

```
INICIO
        Para i Desde 80 Hasta 50 Incrementando de a -1
                Escribir i
        Fin Para
FIN
```

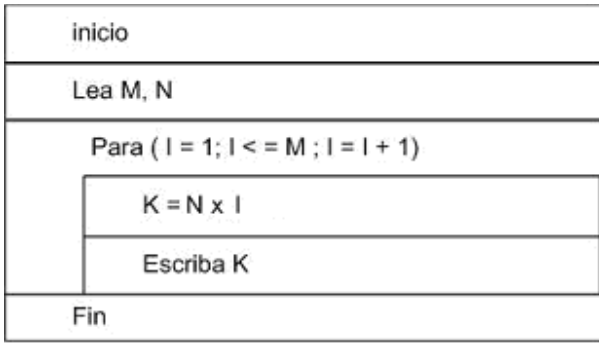
Ejemplo: Diseñar un algoritmo para calcular la tabla de multiplicar del número N desde 1 hasta M.

Así por ejemplo si se le dice que la tabla de multiplicar del 4 hasta 10 debería imprimir:

4
8
12
16
20
24
28
32
36
40

Cuales deben ser entonces las variables de entrada?

Solución:



Como queda este algoritmo traducido en C?

Ejemplo: Diseñar un algoritmo para leer un número entero y determinar si es primo o no.

Solución:

```
INICIO
        Leer X
        Hacer nd = 0
        Para i Desde 1 Hasta X Incrementando de a 1
                Si X % i = 0
                        Hacer nd = nd +1
                Fin Si
        Fin Para
        Si nd > 2
                Escribir "No es primo"
```

```
        De lo contrario
                Escribir "Si es primo"
        Fin Si
FIN
```

Importante: En este caso la variable *nd* actúa como un contador, es decir, que sirve para saber cuantas veces se cumple una condición.

Ejemplo: Diseñar un algoritmo para sumar los números enteros impares entre P y Q.

Solución: Una forma de resolver este problema es la siguiente. Otra podría ser que el indicador de las iteraciones incremente de a una unidad y se pregunte si es impar dentro del FOR.

```
INICIO
        Leer P, Q
        Si (P % 2 = 0)
                Hacer P = P +1
        Fin Si
        Hacer s = 0
        Para i Desde P Hasta Q Incrementando de a 2
                Hacer s = s + 1
        Fin Para
        Escribir "La suma es", s
FIN
```

Importante: En este caso la variable *suma* actúa como un acumulador, es decir, que en ella se van adicionando valores hasta obtener in total.

Ejercicio: En una empresa se tienen N trabajadores, por cada uno de ellos se tienen los siguientes datos: salario básico hora y horas trabajadas en la semana. Si trabaja más de 48 horas las horas extras se pagan con un recargo del 35%. Se debe diseñar un algoritmo para leer los datos de cada trabajador, calcular su salario semanal y el salario total semanal que debe pagar la empresa.

Solución: Primero hay que preguntarse cuales son las variables de entrada? Cuales son las variables de salida?

Sean:

<u>Variables de entrada</u>
N: número de trabajadores
sbh: salario básico hora de un trabajador
hts: horas trabajadas en la semana por un trabajador

<u>Variables de salida</u>
s_trabajador: salario semanal de un trabajador

s_total: salario total semanal que debe pagar la empresa

Para los cálculos hay que tener en cuenta que:
Si se trabajan 48 horas o menos el salario semanal de un trabajardor es sbh*hts.
Pero si se tienen horas extras se les debe adicionar (hts-48)*sbh*0.35
El salario de la empresa se encuentra acumulando los salarios de todos los trabajadores.

Pseudocódigo:

```
INICIO
        Leer N
        Hacer s_total = 0
        Para i Desde 1 Hasta N Incrementando de a 1
                Leer sbh, hts
                Si (hts <= 48)
                        Hacer s_trabajador = hts*sbh
                De lo contrario
                        Hacer s_trabajador = 48*sbh + (hts-48)*sbh*1.35
                Fin Si
                Escribir "Salario semanal trabajador", s_trabajador
                Hacer s_total = s_total + s_trabajador
        Fin Para
        Escribir "Salario total semanal que debe pagar la empresa", s_total
FIN
```