**Comprobando condiciones (1: Si)**

## **Comprobando Condiciones (1: Si)**

En casi cualquier secuencia de instrucciones para un ordenador, será vital poder comprobar si se cumple alguna condición. Una primera forma básica de comprobar condiciones es con la orden "SI". Su uso básico sería

Si condicion Entonces

pasos\_a\_dar\_si\_es\_verdadero

SiNo

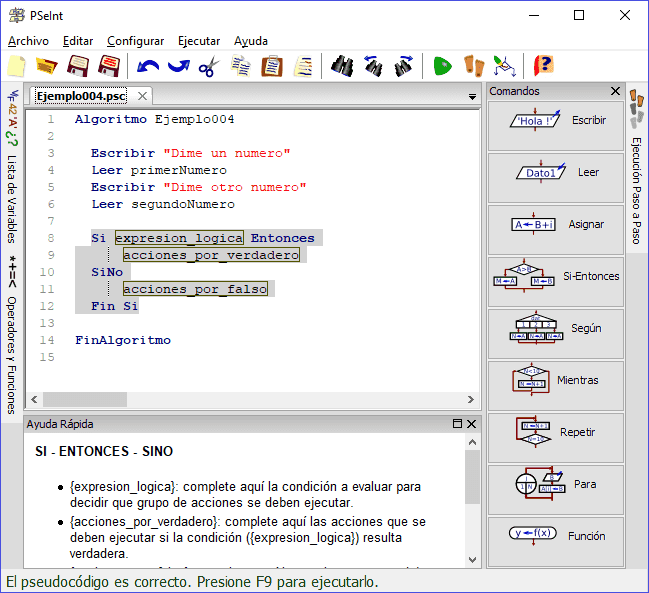
pasos\_a\_dar\_si\_es\_falso

FinSi

El bloque "SiNo" es opcional: podemos optar por no indicar lo que queremos que se haga cuando no se cumpla la condición.

Para ayudarnos a planificar el comportamiento de una secuencia de instrucciones, se suele usar como ayuda los llamados "diagramas de flujo". En estos diagramas, una condición se representa como un rombo, del que salen dos flechas: una para la secuencia de acciones a realizar si se cumple la condición y otra para cuando no se cumple:

Así, si en PSeInt hacemos clic en el icono de la parte derecha que representa la condición SI-ENTONCES, aparece un esqueleto de programa casi completo, para que hagamos los cambios que nos interesen:



Sólo tenemos que escribir la condición que realmente nos interesa, y la serie de pasos que se deben dar si se cumple y si no se cumple, de modo que nuestro programa podría quedar así:

Algoritmo Ejemplo004

Escribir "Dime un numero"

Leer primerNumero

Escribir "Dime otro numero"

Leer segundoNumero

Si primerNumero > segundoNumero Entonces

Escribir "El primero es mayor"

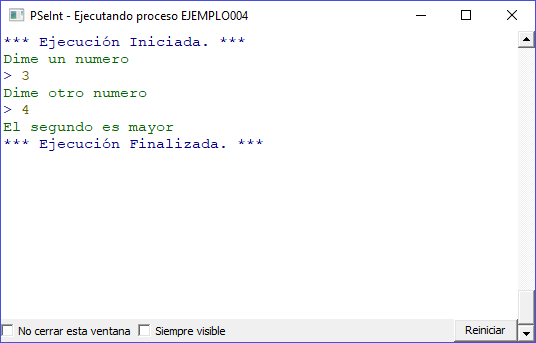
Sino

Escribir "El segundo es mayor"

FinSi

FinAlgoritmo

Su resultado sería éste:



(Nota: si lo piensas con un poco de detenimiento -o lo pruebas-, te darás cuenta de que ese planteamiento no es correcto del todo: si introduces dos números iguales, te dirá que el segundo es el mayor; lo solucionaremos un poco más adelante, en el ejemplo 4c).

En ese ejemplo hemos comparado si un valor es mayor que el otro (>). Los operadores de comparación que tenemos disponibles son:

| **Operador relacional** | **Significado** | **Ejemplo** |
| --- | --- | --- |
| > | Mayor que | 3>2 |
| < | Menor que | 2<3 |
| = | Igual que | 3=3 |
| <= | Menor o igual que | 5 <= 5 |
| >= | Mayor o igual que | 6 >= 5 |
| <> | Distinto de | 6<>5 |

**Ejercicio de repaso propuesto 4.1**: Crea un programa que pida dos números al usuario y responda si son iguales o no lo son.

También es posible comprobar varias condiciones a la vez, para hacer construcciones más complejas, como: "si a es mayor que b y b es mayor que c", o como "si a es igual a 1 o b es igual a 1 ". Los operadores lógicos que de los que disponemos son:

| **Operador lógico** | **Significado** | **Ejemplo** |
| --- | --- | --- |
| & ó Y | Conjunción (y). | (7>4) & (2=1) //falso |
| | ó O | Disyunción (o). | (1=1 | 2=1) //verdadero |
| ~ ó NO | Negación (no). | ~(2<5) //falso |

Un ejemplo de su uso sería:

Algoritmo Ejemplo004b

Escribir "Dime un numero"

Leer primerNumero

Si primerNumero > 0 o primerNumero = 0 Entonces

Escribir "Es mayor o igual que cero"

Sino

Escribir "Es negativo"

FinSi

FinAlgoritmo

Además, podemos encadenar varias condiciones una detrás de otra. Como hemos comentado, el ejemplo 4 fallaba si los dos números son iguales. Por eso, un planteamiento más detallado (y más correcto) sería emplear dos comprobaciones "si", de la siguiente manera:

Algoritmo Ejemplo004c

Escribir "Dime un numero"

Leer primerNumero

Escribir "Dime otro numero"

Leer segundoNumero

Si primerNumero > segundoNumero Entonces

Escribir "El primero es mayor"

Sino

Si primerNumero < segundoNumero Entonces

Escribir "El segundo es mayor"

Sino

Escribir "Son iguales"

FinSi

FinSi

FinAlgoritmo

# Comprobando condiciones (2: Segun)

## **Comprobando Condiciones (2: Segun)**

Es frecuente tener que comprobar más de una condición a la vez, o bien varias condiciones consecutivas. En concreto, un caso especialmente habitual es el de que una variable pueda tomar un valor de entre varios. Por ejemplo, en el sistema de notas escolares español clásico, ciertas notas numéricas tienen "nombres" asociados: un 5 es un aprobado, un 9 y un 10 son sobresaliente, etc. Si queremos hacer un programa que convierta de la nota numérica a su equivalente escrito, podríamos emplear varias órdenes "SI", una tras la otra. Pero en muchos lenguajes de programación (y por tanto, también en muchas variantes de pseudocódigo) existe una alternativa más compacta y más legible: la orden "SEGUN". Esta orden permite hacer unas cosas u otras según el valor que tome una variable. Su uso sería así:

Segun variable Hacer

valor1: pasos\_a\_dar\_si\_es\_el\_valor1

valor2: pasos\_a\_dar\_si\_es\_el\_valor2

valor3: pasos\_a\_dar\_si\_es\_el\_valor3

De Otro Modo:

pasos\_a\_dar\_si\_es\_otro\_valor

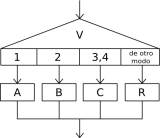
FinSegun

El bloque "De Otro Modo" es opcional: si detallamos todos los valores posibles, no sería necesario utilizarlo.

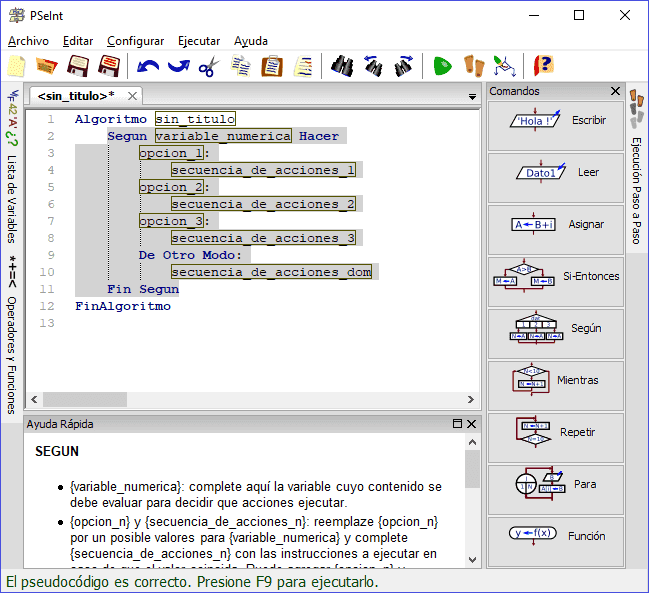
El número de valores que podemos probar es indefinido: no tiene por qué ser 3, sino que pueden ser menos casos o muchos más.

Según la variante de pseudocódigo (o el lenguaje de programación) que empleemos, puede haber restricciones en el tipo de datos que es aceptable. Por ejemplo, en el caso de PseInt, la variable tiene que tener un valor numérico, no puede ser un texto.

Al igual que ocurría con la orden SI, existe un símbolo que podemos usar en los diagramas de flujo para ayudarmos a planificar nuestro programa (aunque este símbolo está menos extendido que el de SI):



Así, si en PseInt hacemos clic en el icono del panel derecho que representa la condición SEGUN, aparece un esqueleto de programa casi completo, para que hagamos los cambios que nos interesen:



Sólo tenemos que escribir la condición que realmente nos interesa, y la serie de pasos a dar si se cumple y si no se cumple, de modo que nuestro programa podría quedar así:

Algoritmo EjemploCasos

Escribir "Introduzca la nota"

Leer nota

Segun nota Hacer

10:

Escribir "Ha obtenido un sobresaliente alto"

9:

Escribir "Ha obtenido un sobresaliente bajo"

8:

Escribir "Ha obtenido un notable alto"

7:

Escribir "Ha obtenido un notable bajo"

6:

Escribir "Ha obtenido un aprobado alto"

5:

Escribir "Ha obtenido un aprobado"

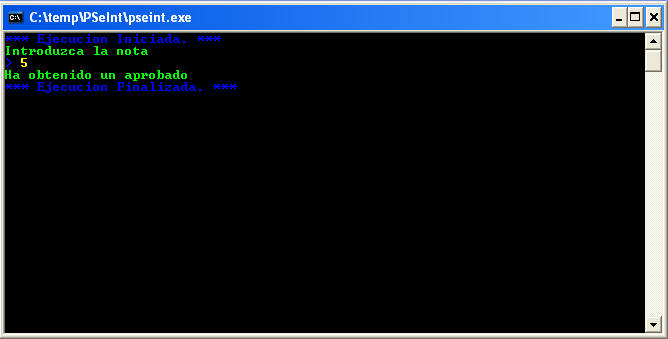
De Otro Modo:

Escribir "Ha suspendido"

FinSegun

FinAlgoritmo

Su resultado sería éste:



Pero no siempre habrá que comprobar condiciones una única vez. Es muy frecuente que haya que hacerlo de forma repetitiva. Por ejemplo, "pedir una contraseña al usuario hasta que sea la correcta", o aún más cercano al mundo real, "pedir una contraseña al usuario hasta que sea la correcta o agote sus intentos". De eso hablaremos en la próxima entrega.

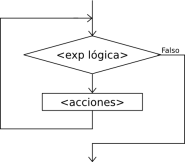
# Condiciones repetitivas (1: Mientras)

## **Condiciones Repetitivas (1: Mientras)**

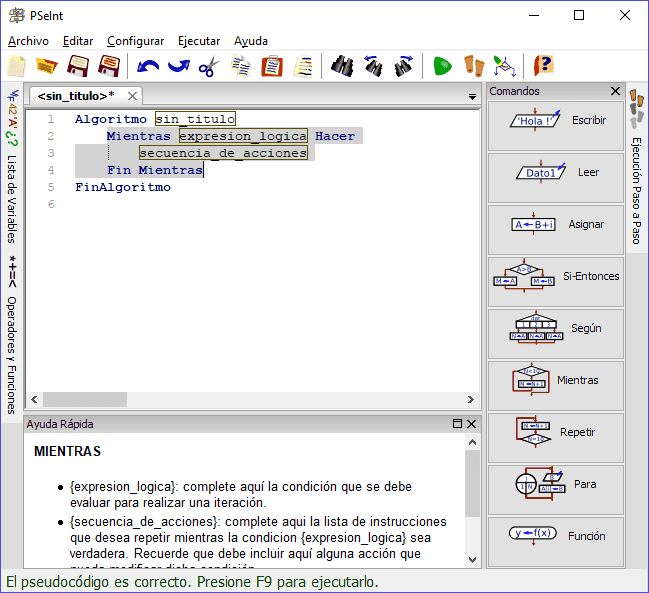
Habitualmente, una condición se deberá comprobar más de una vez. Por ejemplo, una condición de error puede repetirse: el usuario que introduce mal una contraseña por primera vez puede equivocarse también en una segunda ocasión.

Por eso, igual que cualquier lenguaje de programación tiene una orden "si", la gran mayoría de ellos tendrá una orden "mientras", que permite que un fragmento de un programa se repita mientras una cierta condición se siga cumpliendo (por ejemplo, mientras la contraseña que teclee el usuario sea incorrecta, el usuario deberá volver a introducirla).

También existe un símbolo habitual en los diagramas de flujo para representar este tipo de condiciones repetitivas, en las que si se cumple la condición, se realiza una serie de acciones y se vuelve a comprobar la condición, y así sucesivamente hasta que la condición no se cumpla:



Y en el caso de PseInt, ese icono generaría un esqueleto de programa como éste:



Por ejemplo, un programa capaz de sumar muchos números, todos los que el usuario quisiera, y en el que hubiera que escribir "0" para indicar que queremos terminar, podría ser así:

Algoritmo Mientras01

Escribir "Dime un numero"

Leer x

suma <- 0

Mientras x <> 0 Hacer

suma <- suma + x

Escribir "Hasta ahora, la suma es ", suma

Escribir "Dime otro numero"

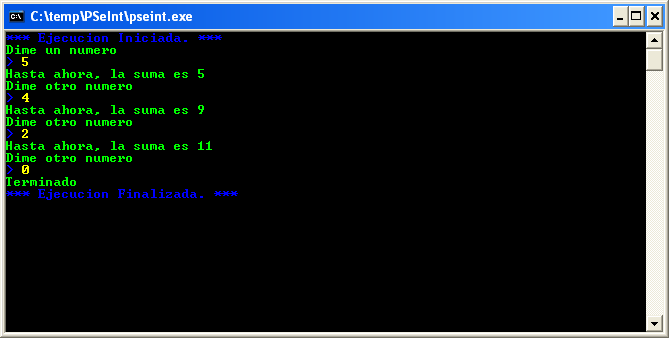
Leer x

FinMientras

Escribir "Terminado"

FinAlgoritmo

Y su ejecución mostraría algo como:



Esta estructuras repetitivas reciben también el nombre de "bucles".

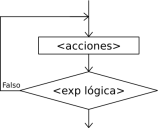
# Condiciones repetitivas (2: Repetir-hasta)

## **Condiciones Repetitivas (2: Repetir-Hasta)**

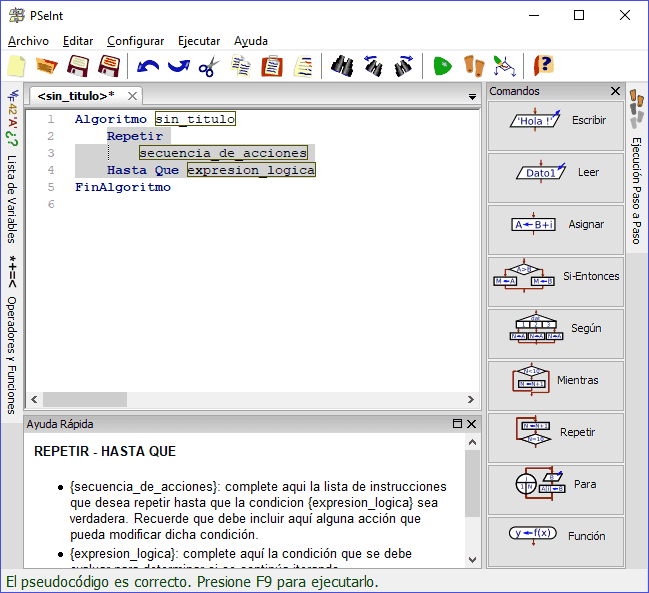
Es también muy frecuente que un bloque de programa repetitivo deba ejecutarse al menos una vez. Por ejemplo, si queremos pedir un dato al usuario, quizá exista algún error y haya que insistir, pero al menos deberemos pedírselo una primera vez.

En estos casos, la estructura "mientras" no es la más adecuada, porque no podemos comprobar antes de pedir el valor. En estos casos (que son muy frecuentes), sería más razonable usar otra estructura de programación en la que la condición se compruebe después de dar ciertos pasos. Esa estructura es "repetir... hasta":

Su representación en un diagrama de flujo sería:



Y en el caso de PSeInt, ese icono generaría un esqueleto de programa como éste:



Por ejemplo, un programa que pida al usuario una clave de acceso, y que no le permita seguir hasta que la introduzca correctamente, se podría hacer así:

Algoritmo Repetir01

Repetir

Escribir "Dime tu clave de acceso"

Leer clave;

Si clave <> 1234 Entonces

Escribir "Clave incorrecta"

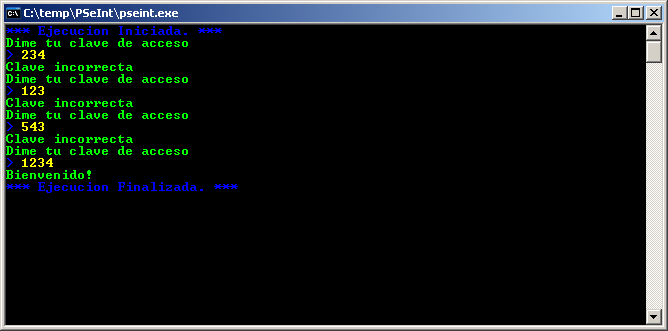
FinSi

Hasta Que clave=1234

Escribir "Bienvenido!"

FinAlgoritmo

Cuyo resultado sería:



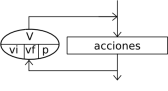
Queda otra forma de repetir fragmentos de programa.

# Repetir un cierto número de veces: Para

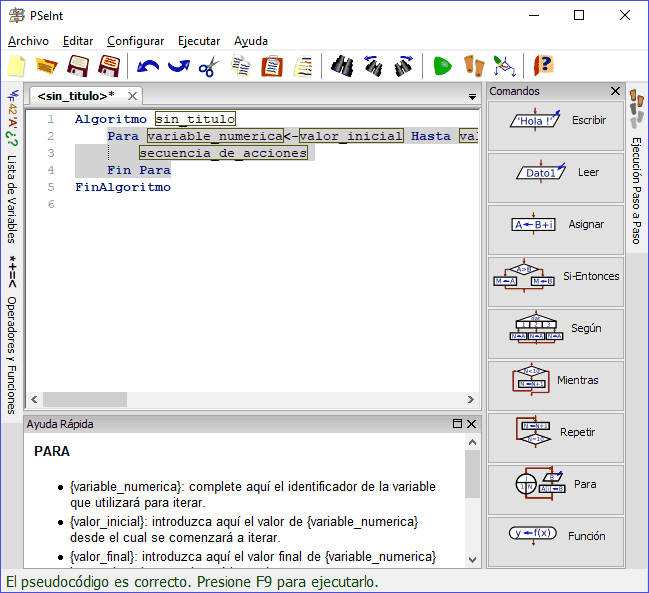
## **Repetir Un Cierto Número De Veces: Para**

En muchas ocasiones, no querremos que algo se repita mientras se cumpla una condición, sino un cierto número de veces. Por ejemplo, para escribir "Hola" 3 veces en pantalla existe una orden más cómoda que la orden "mientras" o la orden "repetir... hasta". Es la orden "para", que hace que una variable tome una serie de valores que se van incrementando. Por ejemplo, una estructura como "para x con valores desde 2 hasta 4" haría que un bloque de programa se repitiera 3 veces. En la primera repetición, la variable "x" tendría el valor 2, en la segunda tendría el valor 3 y en la tercera tendría el valor 4. La sintaxis exacta en PSeInt es: " Para variable <- valorInicial Hasta valorFinal Hacer"

Su representación en un diagrama de flujo sería:



Y en el caso de PSeInt, ese icono generaría un esqueleto de programa como éste:



Por ejemplo, un programa que mostrara los números del 1 al 10, podría ser:

Algoritmo Para01

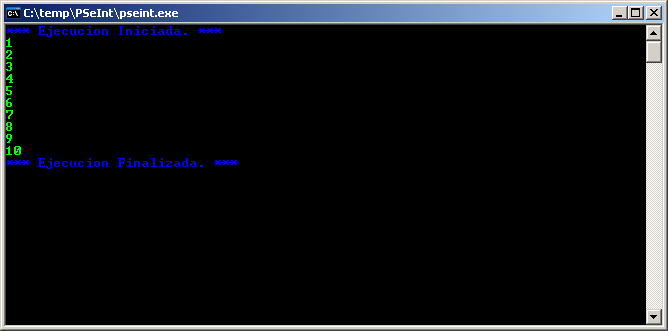
Para x <- 1 Hasta 10 Hacer

Escribir x

FinPara

FinAlgoritmo

Cuyo resultado sería:



Si no queremos avanzar de uno en uno, sino con un incremento distinto, podemos indicar otro tamaño de "paso":

Algoritmo Para02

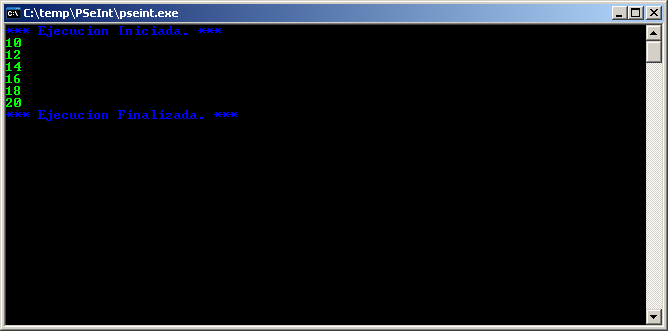
Para x <- 10 Hasta 20 Con Paso 2 Hacer

Escribir x

FinPara

FinAlgoritmo

Y obtendríamos:



Hemos visto casi todas las construcciones que podremos encontrar en los lenguajes de programación convencionales (y, por tanto, al planificarlos usando pseudocódigo). En la próxima entrega veremos alguna otra posibilidad, como el manejo de matrices y las funciones matemáticas, para pasar después a algunos ejemplos completos.