Capítulo 3

ESTRUCTURAS DE PROGRAMACIÓN

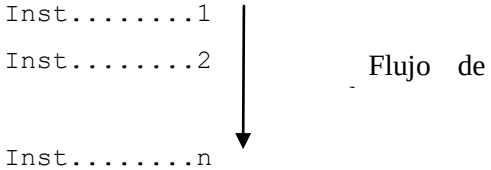
1. INTRODUCCIÓN

Las estructuras de control son usadas por algunas metodologías de programación, estructurada, orientada a objetos, ambas imperativos, para la construcción de algoritmos en la implementación de aplicaciones informáticas. Con estas estructuras, se podrán construir desde los programas más sencillos, hasta los más complicados, razón por la cuál es importante que se comprenda perfectamente su funcionamiento y se sea capaz de utilizarlas correctamente.

2. ESTRUCTURA DE CONTROL SECUENCIAL

En primer lugar, cabe distinguir entre orden físico y orden lógico. Se conoce como **orden físico** la disposición en que están escritas las sentencias o instrucciones y **orden lógico** o **flujo de Control** como el orden en que se realizan las instrucciones durante la ejecución de un programa. El flujo de control puede ser distinto del orden físico de las sentencias. Cuando una instrucción se ejecuta totalmente, el control del procesador pasa a la instrucción siguiente.

Una estructura secuencial está formada por un conjunto de sentencias que se ejecutan una tras otra, y cuyo orden físico y lógico coinciden. Se representa gráficamente con el siguiente formato:



Por ejemplo, instrucciones de declaración de objetos, de asignación, instrucciones de entrada o instrucciones de salida.

EJERCICIO PRÁCTICO DE APLICACIÓN

Ejercicio práctico de aplicación de la Unidad 2

3. ESTRUCTURA DE CONTROL ALTERNATIVA

Esta estructura de programación se conoce también como **estructura condicional**, **de selección** o **de bifurcación**. Se utiliza cuando en determinado punto de un programa se requiere que se elija entre dos o más acciones distintas, en función de alguna condición determinada. El orden físico y lógico no coinciden.

Esta estructura se representa con la instrucción *SI* y su evaluación sólo puede tener resultado verdadero o falso. Existen diversas formas de construir una sentencia *SI*.

3.1. Si - Sino

La estructura de control de selección *SI - SI NO (*o *EN_OTRO_CASO)* es una instrucción que tiene el siguiente formato o sintaxis.

Formato:

```
SI (expresión booleana) [ENTONCES]
        Instrucción A
[SI NO (o EN_OTRO_CASO)]
        Instrucción B
Fin_SI
//Línea siguiente a Fin_SI
```

Donde *expresión boolena* puede ser una expresión booleana simple, por ejemplo, (**A= =B**) o compuesta, por ejemplo, (**(A= =B) Y (C <= D)**).

Una expresión booleana tiene como resultado un valor booleano y está construida con variables booleanas, o con expresiones y operadores relacionales.

Instrucción A o *Instrucción B* puede ser una o varias instrucciones.

En la descripción de la sintaxis de las instrucciones, los corchetes suelen utilizase, habitualmente, para significar que su contenido es opcional. Si en algún caso son utilizados con otro propósito, se advertirá convenientemente al lector.

ENTONCES es opcional, será utilizado o no según nuestro propio estilo de diseño. Es frecuente utilizarlo si se va a codificar con un lenguaje de programación que lo utiliza en su sintaxis. Este no es el caso de Java.

SINO o EN_OTRO_CASO sólo se usará si se prevé realizar alguna tarea de no cumplirse la condición del SI.

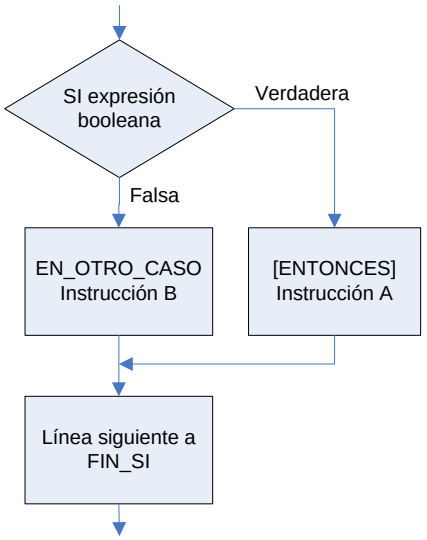


Figura 3.1. Flujo de control en la estructura alternativa

Proceso: ¿Cómo se evalúa? El procesador evalúa la expresión booleana, si se cumple, es decir, si es cierta, hará *Instrucción A* y seguirá el flujo del programa por la sentencia siguiente a Fin_SI. Si no se cumple, es decir, si es falsa, hará *Instrucción B* y, a continuación, seguirá el flujo del programa por la sentencia siguiente a Fin_SI.

En la figura 3.1, se observa la representación gráfica del flujo de control del programa. La flechas del gráfico señalan la dirección del flujo cuando el procesador encuentra una sentencia SI.

Ejemplo 1:

```
.............
.....................
Leer (nacionalidad)
SI (nacionalidad = ='E')
      Escribir("Este señor es español")
Fin_Si
.....................
.............
```

Ejemplo 2:

```
.............
.....................
SI (encontrado = =verdadero)
      Escribir("El alumno está en clase")
```

```
Sino
      Escribir("Ha faltado")
Fin_Si
.....................
.............
```

EJERCICIO PRÁCTICO DE APLICACIÓN

Diseñar un programa que lea del teclado tres números, si el primero es negativo, debe imprimir el producto de los tres y si no lo es, imprimirá la suma.

Nombre del programa: *ProductoSuma*

Análisis: El objetivo del ejercicio es introducir tres números por teclado y realizar operaciones diferentes, según el primero sea o no negativo, por lo que se necesita una estructura alternativa que nos permita comparar y realizar las operaciones oportunas.

Entrada: Tres números enteros.

Salida: Eco de los datos, resultado de la operación multiplicar o de sumar, también enteros y los mensajes que ayuden a la comunicación con el usuario.

Suposiciones: Se considerará que no se dan casos de error en la entrada de datos.

Pseudocódigo generalizado

```
Inicio
      Obtener datos
      Ejecutar operaciones /* Los resultados se imprimirán
                              dentro de este módulo */
FinPP
```

Módulos

```
Obtener datos
      Solicitar tres números por teclado
      Leer los tres números
fin_obtenerDat

Ejecutar operaciones
      Si el primer número es negativo
            Multiplicar
            Imprimir resultado Multiplicación
      Sino
            Sumar
            Imprimir resultado Suma
      Finsi
fin_ejecutarOp
Imprimir resultado Multiplicación
```

```
      Escribir el resultado de la multiplicación
Fin_ImprimirRM

Imprimir resultado Suma
      Escribir el resultado de la Suma
Fin_ImprimirRS
```

Se realiza la inspección ocular del pseudocódigo generalizado y de los módulos, para confirmar que en este nivel de detalle el código responde a las especificaciones propuestas en el análisis. Puede observarse que cada uno de ellos es correcto en unitario y en integración, y realiza exactamente lo que se espera de acuerdo a los requisitos.

Pseudocódigo

```
/*Nombre del programa: ProductoSuma */
ENTORNO:
      VARIABLES:
          entero num1, num2,num3, multiplica, suma
```

Pseudocódigo detallado (o programa principal detallado)

```
<Inicio>
      /*Obtener Datos */
      Escribir("Introduzca tres números enteros") /*Solicitar
                                                   números */
      Leer(num1,num2,num3) /*Leer números */
      /*Fin Obtener Datos*/

      /*ejecutar operaciones */
      SI (num1<0)
            multiplica = num1* num2* num3
            Escribir("Producto de los números: ---",multiplica)
      SINO
            suma=num1+num2+num3
            Escribir("Suma de los números: ---",suma)
      FIN_SI
      /*Fin ejecutar operaciones */

FinPP
```

Observa que los módulos de imprimir resultados se han escrito directamente en detallado `Escribir(...)`, puesto que son sentencias muy simples.

Se pasará la traza al pseudocódigo detallado para verificar su funcionamiento.

Mensaje en pantalla	num1	num2	num3	multiplica	suma
Introduzca tres números	2	1	10		13
Introduzca tres números	-2	1	10	-20	
Introduzca tres números	5	-10	-1		-6

La salida en pantalla para cada una de las ejecuciones anteriores serán las que aparecen a continuación:

```
Suma de los números: 13
Producto de los números: -20
Suma de los números: -6
```

Puede comprobarse que, efectivamente, el algoritmo realiza correctamente la solución del ejercicio.

3.2. Si anidados

No existen restricciones sobre las instrucciones que puede contener una sentencia *SI*, por tanto, puede llevar dentro otro *SI* y este, otro *SI* y así sucesivamente. El único factor que limita el nivel de anidamiento es la legibilidad del código. Cuando se construye una agrupación de este tipo, se dice que se está creando una **estructura de control anidada** cuyo formato es:

```
SI (expresión 1)
      SI (expresión 2)
            SI (expresión 3)
                  Instrucciones A
            SINO
                  Instrucciones B
            FIN_SI
      SINO
            Instrucciones C
      FIN_SI
SINO
      Instrucciones D
FIN_SI
```

Cuando se encuentre ante un problema en el que una condición debe ser comprobada antes de que se evalúe otra condición, es adecuado usar la estructura anidada. Este formato no es el más sencillo de usar, ni el más cómodo, ni siquiera el mejor, pudiendo ser sustituido por varios *SI* con condiciones compuestas e incluso con varios *SI* independientes.

Ejemplo:

Una empresa tiene trabajadores de categorías A, B y C, cobran todos por trimestre un suplemento de 240 €, excepto los de categoría C. De estos, en la sección 1ª están los contratados por días, que cobran un suplemento de 0.5 € por día trabajado

y se les descuenta 30 € por día de baja injustificada. El resto de las secciones de esta categoría cobra 120 € por suplemento.

No se realizará el algoritmo completo para este ejercicio, sino que se construirá sólo el grupo de sentencias que reflejan las condiciones del enunciado.

Observe que todos los trabajadores cobran los 240 € excepto los de categoría C, por tanto, será más fácil construir este *SI* preguntando si se pertenece a esta última categoría.

De igual manera , se procede para el resto de los casos.

```
..............
..............
SI ( cat = = 'c' )
      SI ( secc = = 'p' )
            SI ( dias_baja_inj = ='n' )
                  sup=0.5*diasTrab
            SINO
                  desc=30*numDiasBajaInj
            FIN_SI
      SINO
            sup==120
      FIN_SI
SINO
      sup==240
FIN_SI
..............
```

3.3. Si – Sino - si

Esta construcción del *SI* es una estructura de control apropiada cuando debe realizarse una serie de comparaciones por rangos consecutivos. El formato es:

```
SI (expresión 1)
      Instrucciones A
SINO
      SI (expresión 2)
            Instrucciones B
      SINO
            Instrucciones C
      FIN_SI
FIN_SI
```

EJERCICIO PRÁCTICO DE APLICACIÓN

Realizar un programa para leer de teclado una temperatura y decir el deporte apropiado según los siguientes criterios:

Deporte	Temperatura
Natación	T > 30º
Golf	15º < T <=30º
Tenis	5º < T <= 15º
Esquí	-10º < T <= 5º
Damas	T <= -10º

Nombre del programa: *Deporte*

Análisis: Para comprobar los rangos de las temperaturas se necesita una estructura *si-en otro caso- si* ya que las actividades a realizar están comprendidas en rangos consecutivos.

Entrada: Un entero correspondiente a una temperatura.

Salida: Un mensaje diciendo el deporte apropiado para esa temperatura.

Suposiciones: Se supone que la entrada de datos es correcta.

Pseudocódigo generalizado // del programa principal

```
Inicio
      Obtener datos
      Comprobar tipo de deporte /*Los resultados se presentan
                                 en este módulo */
FinPP
```

Módulos

```
Obtener datos
      Solicitar una temperatura
      Leer temperatura
Fin_Obtener
```

El siguiente módulo no se detalla aquí su código, se hará en el pseudocódigo detallado para no escribirlo dos veces. A continuación solo escribimos el proceso que realiza.

```
Comprobar tipo de deporte
      Proceso: Comparar la temperatura leída con la tabla dada
      Y Escribir en pantalla el deporte apropiado en cada caso.
Fin_Comprobar
```

Al realizar la inspección ocular del pseudocódigo generalizado y de los módulos se observará que, cada uno de ellos, es correcto tanto unitariamente como en integración y realiza exactamente lo que se espera de acuerdo a los requisitos.

Pseudocódigo

```
/*Nombre del programa: Deporte*/
ENTORNO:
      VARIABLES:
            entero temperatura
```

Programa principal detallado

```
<Inicio>

      /*Obtener datos*/
      Escribir("Introduzca una temperatura")
      Leer(temperatura)
      /*Fin Obtener datos*/

      /*Comprobar tipo de deporte*/

      SI (temperatura >30)
         Escribir("Debería practicar Natación")
      SINO
         SI (temperatura >15)
              Escribir("Debería practicar Golf")
         SINO
              SI (temperatura >5)
                   Escribir("Debería practicar Tenis")
              SINO
                   SI (temperatura >-10)
                        Escribir("Debería practicar Esquí")
                   SINO
                        Escribir("Debería practicar Damas")
                   FIN_SI
              FIN_SI
         FIN_SI
      FIN_SI
      /*Fin Comprobar tipo de deporte*/

FinPP
```

A continuación, se realiza la traza.

Mensaje en pantalla	Temperatura	Salida en pantalla
Introduzca una temperatura	18	Debería practicar Golf
Introduzca una temperatura	-2	Debería practicar Esqui
Introduzca una temperatura	-20	Debería practicar Damas
Introduzca una temperatura	35	Debería practicar Natación

EJERCICIO PRÁCTICO DE APLICACIÓN

Realizar un programa que halle el mayor de tres números enteros leídos por teclado y diga si es o no positivo.

Análisis: Como en el caso anterior se usará una estructura *si-en otro caso-si*, que comparará los números para hallar el mayor de ellos. El programa comprobará si el mayor es negativo, verificando si el número en cuestión es menor que 0.

Entrada: Tres números enteros.

Salida: El mayor de los tres números leídos y presentar en pantalla mensaje diciendo si es positivo o no.

Suposiciones: Los números de entrada son válidos, no se hará comprobación de error.

Programa principal generalizado

```
inicio
      Obtener números
      Hallar el mayor
      Comprobar si el mayor es positivo e imprimir en pantalla
FinPP
```

Módulos

No se detallarán separadamente, por ser demasiado simples y haberlo hecho ya en ejemplos anteriores. En su lugar se señalará como comentarios en el programa principal el comienzo y el final de cada módulo que aparece en el generalizado.

Pseudocódigo

```
/*Nombre del programa: Mayor*/
ENTORNO:
      VARIABLES:
          entero num1, num2, num3, mayor
```

Programa principal

```
<Inicio>

     //Obtener números
     Escribir("Introduzca tres números enteros")
     Leer (num1, num2, num3)
     //Fin Obtener números

     //Hallar el mayor
     SI ((num1>num2) y (num1>num3))
       mayor=num1
     SINO
```

```
        SI ((num2>num1) y (num2>num3))
          mayor=num2
        SINO
          mayor=num3
        FIN_SI
      FIN_SI
      //fin hallar el mayor

      //comprobar si el mayor es positivo e imprimir
      SI (mayor > 0)
        Escribir("El número mayor es positivo:  ", mayor)
      SINO
        Escribir("El número mayor es negativo:  ", mayor)
      FIN_SI
      //fin comprobar...

Fin_PP
```

A continuación, se efectua la traza.

Mensaje en pantalla	num1	num2	num3	mayor	Salida en pantalla
Introduzca tres números enteros	2	1	10	10	El número mayor es positivo: 10
Introduzca tres números enteros	-20	10	1	10	El número mayor es positivo: 10
Introduzca tres números enteros	-20	-30	-40	-20	El número mayor es negativo: -20

Observe que con los números de la primera lectura se entra en el último *SINO* del primer *SI*, por tanto, a mayor se le asigna *num3*. Cuando se evalúa el tercer *SI* que aparece en el código, como el valor de mayor, 10, es superior a 0, se obtiene la correspondiente salida en pantalla. Con los datos de la segunda lectura se entra en el segundo *SI*, por lo que, mayor pasa a valer lo que *num2*, al entrar en el último *SI* comprueba que es mayor que 0 y mostrará en pantalla el debido mensaje. Con los datos de la última lectura, se ejecutará el código correspondiente al primer *SI* que aparece en el pseudocódigo, es decir, a mayor se le asigna el valor de *num1*. Al entrar a comparar si es o no positivo, entra en el *SINO* del último *SI* y aparecerá en pantalla el último mensaje.

4. ESTRUCTURA DE CONTROL REPETITIVA, CÍCLICA O ITERATIVA

Es otra forma de hacer que el orden físico de las instrucciones sea diferente del orden lógico en el que se ejecutan. Una estructura repetitiva o **bucle** es una instrucción

formada por una o varias sentencias, conocidas como **cuerpo del bucle**, que se repite un número determinado de veces, mientras se cumpla una condición especificada. Cada vez que se ejecuta el cuerpo del bucle, se dice que se ha producido una **iteración**.

La **salida del bucle** se produce cuando se termina la repetición del cuerpo porque ha dejado de cumplirse la condición de repetición. Cuando esto ocurre, se pasa el control a la sentencia siguiente tras el fin del bucle. La **condición de terminación** es la que provoca la salida de dicho bucle.

El formato o sintaxis de esta estructura aparece a continuación y, como en las estructuras anteriores, *Expresión* puede ser simple o compuesta, e *Instrucciones* puede ser una o varias.

```
Mientras (Expresión) [Hacer]
    Instrucciones  //cuerpo del bucle
Fin_mientras
 //Línea siguiente a Fin_mientras
```

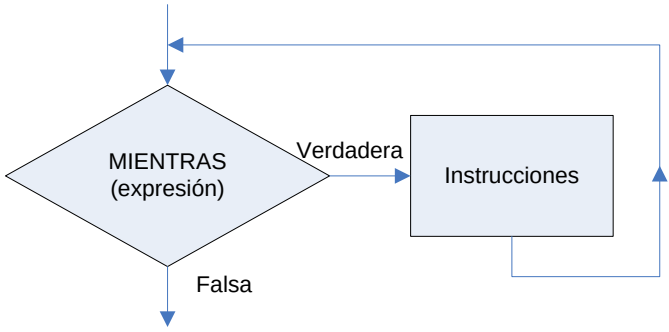


Figura 3.2. Flujo de control en la estructura repetitiva

¿Cómo se realiza el proceso?, el procesador evalúa la expresión, si es cierta se hará el cuerpo del bucle, volverá a evaluar la expresión y así continuará hasta que la evaluación de la expresión sea falsa, en cuyo caso, no volverá a ejecutarse el cuerpo del bucle y seguirá con la instrucción siguiente a *fin_mientras*. Si la expresión es falsa, la primera vez que se evalúa, no ejecutará ninguna vez el cuerpo del bucle e irá directamente a la sentencia siguiente a *fin_mientras*.

Para hacer el programa iterativo en este nivel de conocimientos, se aconseja diseñar el algoritmo como si de una sola ejecución se tratara y poder así centrar la atención en el proceso que se debe realizar. A continuación, se estudiará detenidamente qué sentencias deben añadirse al proceso anterior para hacer que se repita según las especificaciones, es decir, se construye el bucle que será insertado en el lugar correspondiente, con lo que se termina de diseñar el proceso iterativo.

EJERCICIO PRÁCTICO DE APLICACIÓN

Realizar un algoritmo que escriba en pantalla la suma de dos números leídos por teclado y que permita repetir el proceso cuantas veces desee el usuario.

Análisis: Se construirá un programa iterativo que terminará cuando el usuario decida no continuar. En cada iteración realiza un proceso de suma de dos números.

Entrada: Dos números enteros y la respuesta para continuar que será S o N.

Salida: El resultado de la suma, también entero, que se presentará en pantalla y algunos mensajes de refuerzo del código.

Suposiciones: La entrada es válida, no es necesario comprobación de error.

Programa principal generalizado

```
Inicio
     ¿Desea realizar suma de números?
     Mientras la respuesta sea afirmativa
          Obtener datos
          Realizar la suma
          Mostrar en pantalla el resultado de la suma
          ¿Desea realizar suma de números?
     Fin_mientras
FinPP
```

Se efectúa la inspección del pseudocódigo generalizado y se puede confirmar que en este nivel de detalle responde a las especificaciones propuestas en el análisis, suma, nuestra resultados, es iterativo y para cuando se desea.

Módulos

Al igual que en el ejemplo anterior, se incluirán como comentarios el comienzo y final de cada uno de los módulos de los que consta el pseudocódigo generalizado. Tan sólo se detallará el módulo que aparece por primera vez y que controla la posibilidad de realizar nuevamente el proceso.

```
¿Desea realizar suma de números?
      Escribir (¿Quiere realizar la operación de sumar?)
      Leer (respuesta)
Fin-DeseaRS
```

Pseudocódigo

```
/*Nombre del programa: Sumar*/
ENTORNO:
      VARIABLES:
            entero num1, num2, suma
            carácter respuesta
```

Programa principal detallado

```
<Inicio>
     //¿Desea realizar suma de números?
     Escribir (¿Quiere realizar la operación de sumar?)
     Leer (respuesta)
     //Fin DeseaRS
     mientras (respuesta = ='S')
          //Obtener datos
          Escribir("Introduzca dos números enteros")
          Leer (num1, num2)
          //Fin Obtener datos

          //Realizar Suma
          suma = num1+ num2
          //Fin Realizar Suma

          //Mostrar resultado...
          Escribir("El resultado de la suma es:---",suma)
          //Fin Mostrar....

          //¿Desea realizar la suma?
          Escribir (¿Quiere realizar la operación de sumar?)
          Leer (respuesta)
          //Fin DeseaRS
     fin_mientras
Fin_PP
```

Con la traza del algoritmo puede observarse su comportamiento respecto a los requisitos previstos.

Mensaje en pantalla	Respuesta	NUM1	NUM2	suma	Salida en pantalla
¿Quiere realizar la suma?	S	2	1	3	El resultado de la suma es: 3
¿Quiere realizar la suma?	S	-20	10	-10	El resultado de la suma es: -10
¿Quiere realizar la suma?	N				

El programa ha realizado sólo dos iteraciones, realizando la suma de los números de entrada en cada ocasión, como en la última lectura de la variable *Respuesta* se ha leído 'N' al volver a comparar en el *mientras si Respuesta = = 'S'* el resultado es falso, por tanto, pasa a la línea siguiente de *fin_mientras*, que, en este caso, es el final del programa principal.

4.1 Anidamiento de bucles

Como se ha señalado en el apartado anterior un bucle *mientras* es una instrucción, por tanto, también podrá ir dentro de cualquier otra instrucción, por ejemplo, otro bucle *mientras*, y este a su vez dentro de otro y, así, sucesivamente. Se dice que se construye una **estructura repetitiva anidada**.

El nivel de profundidad en el anidamiento lo determina la semántica del ejercicio, pero debe haber un compromiso razonable entre éste nivel, y la legibilidad y comprensión del código generado, lo cual es responsabilidad del programador..

El formato de la estructura anidada en tres niveles es el siguiente:

```
mientras (expresión 1) [hacer]
    mientras(expresión 2) [hacer]
        mientras(expresión 3) [hacer]
            Instrucciones
        fin_mientras
    fin_mientras
fin_mientras
```

Para construir una estructura iterativa anidada, se aconseja comenzar construyendo el bucle exterior, abstrayéndonos de los demás, para después seguir realizando los más internos, o viceversa, según se sienta el propio programador más cómodo en la ejecución del diseño.

5. CONTADORES Y ACUMULADORES

Se tratan a continuación dos tipos de variables utilizadas a menudo en los procesos iterativos que tienen características muy particulares: contadores y acumuladores

5.1 Contadores

Un **contador** es una variable que **se incrementa o decrementa de forma constante** cada vez que se realiza la instrucción que lo contiene. Estas variables se utilizan para contar las veces que ocurre un determinado suceso o para controlar la ejecución de un bucle que se realiza un número determinado de veces.

Por ejemplo, el marcador de un campo de fútbol.

INSTANTE	LOCAL	VISITANTE
0 (Antes de comenzar el partido)	0	0
A (Un momento cualquiera)	2	1
B (Después de marcar otro gol el Local)	3	1

Las casillas LOCAL y VISITANTE almacenarán los goles que van sumando los dos equipos. Antes de comenzar el partido, ambas casillas han sido colocadas al valor 0, pues si permanecen con el resultado del partido anteriormente jugado, no sería correcto que se le fueran sumando a este resultado los goles que se van produciendo en el partido actual.

Todo contador debe tomar un valor inicial antes de ser usado, pues de no hacerlo la constante se sumará a la basura que hubiera en la dirección de memoria donde se almacenan sus valores. Esta operación se llama **inicialización**.

El formato para el uso de una variable que actuará de contador es el siguiente:

Variable_Contador = Variable_Contador + (o -) constante

El procesador realiza la operación a la derecha del símbolo de asignación y el resultado lo almacena en la variable que se encuentra a la izquierda del mismo. Esta operación se llama **actualización** del contador.

La constante puede ser de cualquier tipo de datos, aunque lo razonable es que sea del mismo tipo que la variable contadora.

EJERCICIO PRÁCTICO DE APLICACIÓN

Diseñar un algoritmo que lea números de teclado mientras se desee y cuente cuántos de ellos son pares y cuantos negativos.

Análisis: El algoritmo diseñado debe ser iterativo, acabará cuando el usuario responda 'N'a la pregunta ¿Desea seguir?. Se contarán los números pares comprobando si el resto de dividir entre 2 es cero y se contarán los negativos verificando si el número en cuestión es menor que 0.

Entrada: Números enteros y la respuesta para continuar que será S o N.

Salida: El resultado de contar cuántos de los números leídos son pares y el resultado de contar cuántos son negativos, y algunos mensajes de refuerzo del código.

Suposiciones: Se supone que la entrada de datos es válida, por tanto no hay que añadir código para comprobación de error.

Programa principal generalizado

```
Inicio
      ¿Desea seguir?
      Mientras la respuesta sea afirmativa
            Obtener dato
            Realizar operaciones
            ¿Desea seguir?
      Fin_mientras
      Mostrar Resultados
FinPP
```

Módulos

Se detallará sólo el módulo de *Realizar operaciones*. Los demás son similares a los aparecidos en ejemplos anteriores por lo que aparecen como comentarios en el pseudocódigo detallado.

```
Realizar operaciones
      Calcular y contar Par
      Calcular y contar Negativo
FinRO  //Fin Realizar Operaciones
```

Los módulos **Calcular y contar Par y Calcular y contar Negativo**, también aparecerán en comentarios en el pseudocódigo detallado.

Se efectúa la inspección de código del pseudocódigo generalizado y de los módulos, que como puede observarse en este caso, aparecen en un segundo nivel. Se puede confirmar que responden a las especificaciones requeridas, cuenta pares, cuenta negativos, es iterativo, para cuando quiera el usuario y, antes de terminar, presenta el resultado.

Pseudocódigo

```
/*Nombre del programa: ParesYNegativos*/
ENTORNO:
 VARIABLES: //Inicializamos los contadores
       entero num, contPares = 0, contNegativos = 0
       carácter respuesta
```

Programa principal detallado

```
<Inicio>
      //¿Desea seguir?
      Escribir (¿Desea seguir ejecutando el programa?)
      Leer (respuesta)
      //Fin DeseaS

      mientras (respuesta = ='S')
            //Obtener dato
            Escribir("Introduzca un número entero")
            Leer (num)
            //Fin Obtener dato
            //Realizar operaciones
            //calcular y contar par
            SI (num MOD 2 = = 0)
                  contPares = contPares + 1
            FINSI

            //calcular y contar negativos
            SI (num < 0)
                  contNegativos = contNegativos + 1
```

```
            FINSI
            //¿Desea seguir?
            Escribir (¿Desea seguir ejecutando el programa?)
            Leer (respuesta)
            //Fin DeseaS
      fin_mientras
      //Mostrar resultado...
      Escribir("La cantidad de números pares es ---",contPares)
      Escribir("La cantidad de números negativos es ---
               ",contNegativos)
      //Fin Mostrar....

Fin_PP
```

Realizando la traza del algoritmo puede advertir que su comportamiento respecto a los requisitos previstos es correcto.

Mensaje en pantalla	Respuesta	NUM	contpares	contnegativos
			0	0
¿Desea seguir ejecutando el programa?	S	6	1	
¿Desea seguir ejecutando el programa?	S	-20	2	1
¿Desea seguir ejecutando el programa?	S	3		
¿Desea seguir ejecutando el programa?	S	-5		2
¿Desea seguir ejecutando el programa?	N			

La salida en pantalla es la siguiente:

```
La cantidad de números leídos pares es 2
La cantidad de números leídos negativos es 2
```

El programa ha realizado cuatro iteraciones leyendo un número en cada ocasión y actualizando los dos contadores en el caso correspondiente. Cuando se ha leído 'N' ,como **Respuesta** pasa la ejecución del programa a la línea siguiente de *fin_mientras*, a continuación, se muestran en pantalla los mensajes de salida.

5.2 Acumuladores

Un **acumulador** es una variable que nos permite guardar un valor que **se incrementa o decrementa de forma variable** durante el proceso. Por ejemplo, sacar o meter dinero en una cartilla de ahorros.

Formato:

Variable_Acumulador = Variable_Acumulador + (ó -, *, :)Variable

El procesador realiza la operación a la derecha del símbolo de asignación y el resultado lo almacena en la variable que se encuentra a la izquierda del mismo.

Los acumuladores deben tomar un valor antes de ser usados por primera vez.

Se tendrán en cuenta los dos casos siguientes:

a) Cuando el acumulador debe variar por **suma** de sucesivas cantidades variables, deberá ser inicializado a 0. En caso de no inicializarlos a 0, se irán acumulando las cantidades sobre la basura que hubiera en esa zona de memoria.

Inicialización	`Var_Acumulador = 0`
Actualización	`Var_Acumulador = Var_Acumulador + Variable`

EJERCICIO PRÁCTICO DE APLICACIÓN

Ejemplo: Diseñar un algoritmo que sume los 10 primeros números naturales que sean pares (2,4,6...20).

Análisis: El algoritmo diseñado debe ser iterativo, acabará cuando se hayan procesado los diez primeros números naturales pares.

Entrada: No existe entrada de datos.

Salida: El resultado de sumar los 10 primeros números naturales pares.

Suposiciones: Ninguna.

Programa principal generalizado

```
Inicio
      Inicializar numero y acumulador
      Mientras Numero no haya llegado a 20
            Realizar Suma
            Actualizar Número
      Fin_mientras
      Mostrar Resultados
FinPP
```

Los módulos se especifican sobre el pseudocódigo detallado.

Pseudocódigo

```
/*Nombre del programa: DiezPares*/
ENTORNO:
      VARIABLES:
            entero num = 2, suma = 0 /*Inicializar
                  número y  acumulador */
```

Programa principal detallado

```
<Inicio>
      mientras (num < = 20)

            // Realizar Suma
            suma = suma + num
            //Fin Realizar Suma

            // Actualizar Número
            num = num+2
            //Fin Actualizar...
      fin_mientras

      //Mostrar resultado...
                  Escribir("La suma de los 10 primeros
                        números pares es ---",suma)
      //Fin Mostrar....
Fin_PP
```

Al realizar la traza del algoritmo se verifica que realmente realiza lo que se espera.

num	suma
2	0
	2
4	6
6	12
8	20
10	30
12	42
14	56
16	72
18	90
20	110
22	

La salida en pantalla es:

```
La suma de los 10 primeros números pares es 110
```

El programa ha realizado diez iteraciones para procesar precisamente los diez primeros números naturales pares. Cuando *num* se actualiza a 22, no se

entra en el bucle y pasa la ejecución del programa a la línea siguiente a *fin_mientras*. A continuación, se visualiza en pantalla el mensaje correspondiente a la suma.

b) En aquellos otros casos en que el acumulador debe variar por producto de sucesivas y variables cantidades, deberá ser inicializado a 1. Como en el caso anterior, si no se inicializa, se irán multiplicando las cantidades sobre la basura contenida en esa zona de memoria.

Inicialización	`Var_Acumulador = 1`
Actualización	`Var_Acumulador = Var_Acumulador * Variable`

EJERCICIO PRÁCTICO DE APLICACIÓN

Hacer un algoritmo que multiplique los 5 primeros números impares naturales (1, 3, 5, 7, 9).

Análisis: El algoritmo diseñado será repetitivo, acabará cuando se hayan procesado los cinco primeros números naturales impares.

Entrada: No existe entrada de datos.

Salida: El resultado de multiplicar los 5 primeros números naturales impares.

Suposiciones: Ninguna.

Programa principal generalizado

```
Inicio
      Inicializar Numero y Acumulador Multiplicativo
      Mientras Numero no haya llegado a 9
            Realizar Producto
            Actualizar Número
      Fin_mientras
      Mostrar Resultados
FinPP
```

Los módulos aparecen como comentarios sobre el pseudocódigo detallado.

Pseudocódigo

```
/*Nombre del programa: CincoImpares*/
ENTORNO:
      VARIABLES: /*Inicializar número y  acumulador
                  multiplicativo */
            entero num = 1, producto = 1
```

Programa principal detallado

```
<Inicio>
      mientras (NUM < = 9)
            // Realizar Producto
            producto = producto * num
            //Fin Realizar producto

            // Actualizar Número
            num = num+2
            //Fin Actualizar...

      fin_mientras

      //Mostrar resultado...
      Escribir("El producto de los 5 primeros números impares
                es ---",Producto)
      //Fin Mostrar....

Fin_PP
```

Puede comprobarse con el seguimiento de la traza del algoritmo que funciona de acuerdo a las especificaciones del problema.

num	producto
1	0
	1
3	3
5	15
7	105
9	945
11	

La salida en pantalla es:

```
El producto de los 5 primeros números impares es 945
```

El programa ha realizado cinco iteraciones procesando los cinco primeros números naturales impares. Cuando *num* se actualiza a 11, no se entra en el bucle y pasa la ejecución del programa a la línea siguiente a *fin_mientras*, para terminar imprimiendo en pantalla el producto.

6. DATO CENTINELA

Un **dato centinela** es un valor particular de una variable que no pertenece al rango de valores admitidos como válidos para esa variable y que va a permitir finalizar un proceso.

Por ejemplo, si se debe hacer un proceso consistente en leer números positivos, un dato centinela podría ser -1, si se trata de procesar los diez primeros números pares, el dato centinela puede ser el 22, como ocurría en el ejemplo anterior, si la tarea es leer nombres de alumnos, se puede elegir como dato centinela "último", "FIN"o bien, "*****".

Observe, que el dato centinela en los ejemplos anteriores no es un valor normal de la variable, sino que, es algo que nunca pertenecerá al rango de valores posibles de los datos.

EJERCICIO PRÁCTICO DE APLICACIÓN

Diseñar un algoritmo que sume números pares positivos cualesquiera hasta que se lea -1.

Análisis: El algoritmo será iterativo, acabará cuando se introduzca desde teclado -1.

Entrada: Números enteros positivos.

Salida: El resultado de sumar sólo aquellos números introducidos que sean pares.

Suposiciones: Consideramos que la entrada de datos es válida. No se realizará, por tanto, comprobación de error.

Programa principal generalizado

```
Inicio
       Inicializar Suma
       Obtener Numero
       Mientras Numero sea distinto de -1
         Comprobar Par y Sumar
         Obtener Número
       Fin_mientras
       Mostrar Resultados
FinPP
```

Pseudocódigo

```
/*Nombre del programa: SumaPares*/
ENTORNO:
      VARIABLES:
            entero num, suma = 0 //Inicializar Suma
```

Programa principal detallado

```
<Inicio>
      //Obtener Numero
      Escribir("Introduzca un número entero positivo, -1 para
                terminar")
      Leer (NUM)
```

```
   //Fin Obtener Numero

   mientras (num <> -1) //Mientras número sea distinto de -1
      //comprobar par y sumar
      SI (num MOD 2 = = 0)
          suma = suma + num
      FINSI
      //Fin comprobar par y sumar

      //Obtener Numero
      Escribir("Introduzca un número entero positivo, -1 para
                terminar)")
      Leer (num)
      //Fin Obtener Numero
   fin_mientras

   //Mostrar resultado...
   Escribir("Suma de los números pares leídos: ---",suma)
   //Fin Mostrar....

Fin_PP
```

Observe el comportamiento del algoritmo respecto a los requisitos previstos al seguir la traza con las entradas que se detallan a continuación.

Mensaje en pantalla	num	suma
		0
Introduzca un número entero positivo, -1 para terminar	6	6
Introduzca un número entero positivo, -1 para terminar	13	
Introduzca un número entero positivo, -1 para terminar	28	34
Introduzca un número entero positivo, -1 para terminar	7	
Introduzca un número entero positivo, -1 para terminar	205	
Introduzca un número entero positivo, -1 para terminar	-1	

La salida en pantalla es la siguiente:

```
Suma de los números pares leídos: 34
```

El programa ha realizado cinco iteraciones procesando números naturales introducidos, comprobando si son pares en cuyo caso se suman. Cuando *num* se actualiza a -1, no se entra en el bucle y pasa la ejecución del programa a la línea siguiente a *fin_mientras*. El programa acaba imprimiendo en pantalla la suma de los números pares introducidos.

7. SWITCH, FLAGS O INDICADORES

Los **indicadores** también llamados **banderas** (**flags**) o **interruptores** (**switch**), son variables booleanas, por tanto, toman sólo dos valores "verdad o falso". Un indicador se utiliza para controlar el flujo lógico de un programa y permite variar la secuencia de ejecución dependiendo del valor que tenga en cada instante. Al valor se le suele llamar también **estado**, debido a que un dispositivo físico interruptor (por ejemplo, el interruptor de la luz) puede hallarse en estado encendido o apagado.

Se usan para:

- Saber si el programa ha pasado por un determinado punto preguntando por su estado.
- Salir de un ciclo cuando el switch toma un determinado valor.
- Ejecutar una u otra acción dependiendo de su estado.

EJERCICIO PRÁCTICO DE APLICACIÓN

Diseñar un algoritmo que sume números pares positivos leídos por teclado, hasta que se lea un número negativo cualquiera. Usar para controlar la entrada de datos una variable switch.

Análisis: El algoritmo será iterativo. **Se usará un indicador (*Negativo*), inicializado a "falso" que cambiará de valor cuando se lea desde teclado un número negativo**. ¡OJO!

Entrada: Números enteros.

Salida: El resultado de sumar sólo aquellos números introducidos que sean pares.

Suposiciones: Estimamos que la entrada de datos es válida, por lo que no se realizará tratamiento de error.

Programa principal generalizado

```
Inicio
      Inicializar Indicador y Suma
      Mientras Indicador de negativo sea falso
            Obtener Número
            Comprobar y Sumar
      Fin_mientras
      Mostrar Resultados
FinPP
```

Pseudocódigo

```
/*Nombre del programa: SumaParesIndicador*/
ENTORNO:
```

```
VARIABLES:
      entero num, suma = 0 //Inicializar Suma
      Lógico negativo = falso //Inicializar indicador
      /*Nota: si quiero hacerlo así hay que evitar que el
      proceso se haga si leo un negativo. Es decir,
      tendré que hacer la lectura del número dentro y
      controlar el suceso. Si no es así, habrá que leer
      el número antes del mientras y controlar el suceso
      y la inicialización del indicador.*/

      /* Discutir ambos diseños y comprobar las
      diferencias existentes*/
```

Programa principal detallado

```
<Inicio>

    mientras (negativo = = falso)
        //Obtener Numero
        Escribir("Introduzca un número entero cualquiera,
                  negativo para terminar")
        Leer (num)
        //Fin Obtener Numero

        // Comprobar y sumar
        SI (num < 0)
          negativo = verdadero
        SINO
          SI (num MOD 2 = = 0)
            suma = suma + num
          FINSI
        FINSI
        //Fin Comprobar...

    fin_mientras

    //Mostrar resultado...
    Escribir("La suma de los números pares positivos leídos es
              ---",suma)
    //Fin Mostrar....

Fin_PP
```

Con el juego de ensayo de los datos que se utilizan en esta traza, puede comprobarse que el algoritmo responde a los requisitos del ejercicio. Obsérvese el cambio de estado del indicador.

Mensaje en pantalla	negativo	num	suma
	falso		0
Introduzca un número entero cualquiera, negativo para terminar		6	6
Introduzca un número entero cualquiera, negativo para terminar		13	
Introduzca un número entero cualquiera, negativo para terminar		28	34
Introduzca un número entero cualquiera, negativo para terminar		7	
Introduzca un número entero cualquiera, negativo para terminar		204	238
Introduzca un número entero cualquiera, negativo para terminar		-27	
	verdadero		

La salida en pantalla es la siguiente:

```
La suma de los números pares positivos leídos es 238
```

El programa ha realizado seis iteraciones procesando los números leidos, comprobando si son negativos, en cuyo caso el indicador cambia de estado y si son pares en cuyo caso se suman. Cuando *num* vale -27, el indicador se actualiza a "verdadero", por tanto, no se entra en el bucle y pasa la ejecución del programa a la línea siguiente a *fin_mientras*. Al final, se imprime en pantalla la suma de los números pares positivos introducidos por teclado.

8. TIPOS DE BUCLES

Como se ha podido observar en los ejemplos anteriores, el control para la ejecución de un bucle recae en alguno de los objetos que forman parte de la expresión correspondiente a la condición de repetición de dicho bucle. Estos objetos suelen ser una o más variables que se conocen como **Variables de Control del Bucle**, **VCB**.

En la resolución de problemas encontrará, en la gran mayoría de los casos, dos tipos de bucles que se repiten frecuentemente:

- **Bucles controlados por contador**. Se repite un número conocido de veces.
- **Bucles controlados por** sucesos. Se repite hasta que ocurre algo dentro del cuerpo del bucle que provoca la terminación del proceso de iteración, es decir, indica que se debe salir del bucle.

Por ejemplo, si nos encontramos realizando un pastel y en la receta aparece "batir la mezcla 100 veces ", estamos ejecutando un bucle controlado por contador. Sin embargo, si la receta dice "batir la mezcla hasta que tenga textura de almíbar" estaremos ejecutando un bucle controlado por suceso, en este caso el suceso es *tener textura de almíbar*.

8.1 Bucles controlados por contador

Un bucle controlado por contador tiene las siguientes características:

- Se ejecuta un número determinado de veces.
- Hace uso de una variable de tipo contador para control del bucle.

Hay tres operaciones que se realizan siempre en este tipo de bucles:

- Inicializar el contador o VCB.
- Evaluar el contador o VCB, por comparación con el número de iteraciones especificado.
- Variar o actualizar el contador o VCB (Incrementar o decrementar, no necesariamente de 1 en 1).

Formato	Ejemplo
	`CONSTANTE` `  N = 10`
Inicializar la VCB	`contador = 1 /*Inicializar VCB*/`
.....	`.............`
Mientras (Expresión para Comparar VCB)	`Mientras (contador <= N)` `/* Comparar VCB*/`
.....	`...............`
.....	`...............`
Expresión para Variar la VCB	`contador = contador + 1` `/*Incrementar o decrementar VCB*/`
Fin_mientras	`Fin_mientras`

Hay que prestar especial atención a lo siguiente:

- Se debe inicializar correctamente la VCB para que entre en el bucle la primera vez.
- La VCB debe variar dentro, en el cuerpo del bucle, para propiciar la condición de salida, de lo contrario se provocarán los temidos bucles infinitos.
- Es importante estudiar cuántas veces se ejecuta el bucle y comprobar que se cumple el número de iteraciones previsto. Esto es independiente del valor al que se inicialice la VCB. Por ejemplo, si *Contador* se inicializa a 1 y se requiere que el proceso se repita *N* veces, la condición de repetición será *Contador* < = *N*. Pero, si se hubiera inicializado *Contador* a 0, la condición para que el proceso se itere *N* veces será *Contador* < *N*.

EJERCICIO PRÁCTICO DE APLICACIÓN

Diseñar un algoritmo para tomar la temperatura exterior leyéndola durante un periodo de 24 horas. Debe hallarse la temperatura más alta y la más baja del día a partir de estos datos.

Análisis: El proceso de leer la temperatura debe repetirse cada hora, por tanto, se necesita un bucle controlado por contador que se moverá desde las 0 hasta las 23 horas leyendo 24 temperaturas. El proceso acabará cuando el contador llegue a 24. A medida que se leen, se deben ir comparando las temperaturas para hallar la más alta y la más baja.

Entrada: Números enteros.

Salida: La temperatura más alta y la más baja, y algunos mensajes de refuerzo del código.

Suposiciones: Se considera válida la entrada de datos, por tanto no hay que añadir código para comprobación de error.

Programa principal generalizado

```
Inicio
      Inicializar contador
      Mientras contador menor que 24
            Obtener Temperatura
            Calcular mayor
            Calcular menor
            Actualizar contador
      Fin_mientras
      Mostrar Resultados
FinPP
```

Módulos

Los módulos se especifican dentro del pseudocódigo detallado.

Se realiza la inspección de código del pseudocódigo generalizado. Puede observarse que responden a las especificaciones requeridas, lee temperaturas cada hora del día hasta 24, calcula la mayor, la menor y antes de terminar presenta el resultado en pantalla.

Pseudocódigo

```
     /*Nombre del programa: Temperatura*/
     ENTORNO:
      VARIABLES: //Inicializar el contador, cont es la VCB
          entero temperatura, maxima = 0, minima = 0, cont = 0
```

Programa principal detallado

```
<Inicio>
      mientras (cont < 24) /*Condición de entrada al bucle
                              controlado por contador */
          //Obtener Temperatura
          Escribir("Introduzca la temperatura")
          Leer (temperatura)
          //Fin Obtener Temperatura

          //calcular mayor
          SI (temperatura > maxima)
            maxima = Temperatura
          FINSI
          //Fin calcular mayor

          //calcular menor
          SI (temperatura < minima)
            minima = temperatura
          FINSI
          //fin calcular menor

           cont = cont +1 //Actualización del contador
     fin_mientras

     //Mostrar resultado...
     Escribir("La temperatura más alta es ---",maxima)
     Escribir("La temperatura más baja es ---",minima)
     //Fin Mostrar....

Fin_PP
```

La traza que se muestra a continuación permite verificar el código del algoritmo descrito anteriormente.

Mensaje en pantalla	temperatura	maxima	minima	cont
		0	0	0
Introduzca la temperatura	13	13		1
Introduzca la temperatura	-5		-5	2
Introduzca la temperatura	4			3
Introduzca la temperatura	18	18		4
Introduzca la temperatura	-4			5
Introduzca la temperatura	-10		-10	6
.....	.....	.....	.....	...
Introduzca la temperatura	-6			23
				24

La salida en pantalla, si no se hubiera introducido ninguna temperatura mayor que 18 ni menor que -10, sería la siguiente:

```
La temperatura más alta es 18
La temperatura más baja es -10
```

El programa realiza 24 iteraciones leyendo una temperatura en cada ocasión y actualizando la máxima o la mínima si hubiera lugar y el contador. Cuando la VCB llega a valer 24, no se cumple la condición de entrada y pasa la ejecución del programa a la línea siguiente de *fin_mientras*. Por último, se presentan los resultados en pantalla.

8.2 Bucles controlados por sucesos

En estos tipos de bucles la condición de terminación depende de algún suceso que ocurre durante la ejecución del cuerpo del bucle.

Hay varios tipos de bucles controlados por sucesos:

- Controlados por centinelas.
- Controlados por switch o indicador.
- Controlados por fin de fichero. Estos se verán en el capítulo dedicado a ficheros.

A) BUCLES CONTROLADOS POR CENTINELAS

Los bucles se utilizan a menudo para leer y procesar grandes listas de datos y es frecuente utilizar un centinela como señal de que no hay que procesar ningún dato más, es decir, la lectura del centinela es el suceso que controla el proceso de iteración. La variable de control de bucle es un dato centinela.

Hay tres operaciones que se deben hacer siempre en este tipo de bucle:

- **Lectura anticipada**. La VCB debe leerse antes de entrar en el bucle, lo que hará que la variable contenga un valor válido y real antes de la comparación.
- **Evaluar la VCB**, por comparación de su valor con el dato centinela.
- **Lectura final**. Antes de cada repetición debe leerse el centinela, para que al realizar la comparación se evalúe la condición de iteración nuevamente. Esto ocurre al final del cuerpo del bucle, por lo que se llama lectura final.

Formato:

```
Lectura de datos //lectura anticipada
```

```
Mientras(Evaluación de la VCB) /*¿El dato leído es
                                el centinela? */
        ..........
        <proceso>
        ............
        Lectura de datos //lectura final
    Fin_mientras
```

El proceso sucede del siguiente modo: se leen los datos, se compara el valor de la VCB con el dato centinela (mediante *expresión*), si el resultado es cierto, es decir, el dato leído no es el dato centinela, se procesa el cuerpo del bucle, se leen de nuevo los datos, se vuelve a comparar, se procesa y, así, sucesivamente. Cuando el valor de la VCB es el dato centinela, es decir, cuando se ha leído dicho valor, se acaba el bucle.

B) BUCLES CONTROLADOS POR INDICADOR

En estos tipos de bucles se utiliza como VCB una variable booleana que registra si se ha producido o no el suceso que controla el proceso.

Las operaciones que se realizan en estos bucles son:

- **Inicializar el indicador o VCB**. El indicador no puede leerse puesto que se trata de una variable booleana.
- **Evaluar el indicador o VCB**, comparando su valor según la condición prevista para repetir el bucle.
- **Actualizar el indicador o VCB**. Cambio de estado del indicador causado por un suceso en particular, que será lo que provoque la terminación del proceso iterativo.

  Ten en cuenta que la evaluación del suceso deberá hacerse mediante una condición, por lo que esto llevará consigo una estructura de control de flujo condicional, es decir un ***SI***.

Formato:

```
Inicializar indicador /*Para que entre en el bucle la
                      primera vez*/
Mientras (Evaluación del indicador)
      ....... // Cuerpo del bucle
      Actualización del indicador
Fin_mientras
```

EJERCICIO PRÁCTICO DE APLICACIÓN

Realizar un programa para leer de teclado números enteros positivos y sumar los múltiplos de 5. La entrada acaba al introducir un número negativo. Realizar el control del bucle con un indicador.

Análisis: Se construirá una estructura iterativa controlada por suceso, cuya VCB será un indicador, que tomará valor "cierto" si el número leído es positivo y "falso", en caso contrario.

Se aconseja al alumno que realice el ejercicio con estructura repetitiva controlada por un dato centinela, -1 por ejemplo, para observar las diferencias en el código entre el ejercicio resuelto con indicador y el resuelto mediante un centinela como VCB.

Entrada: Números enteros.

Salida: El resultado de sumar los datos de entrada que sean múltiplos de 5 y algunos mensajes de refuerzo de código.

Suposiciones: No hay comprobación de error, la entrada es válida.

Programa principal generalizado

```
Inicio
      Inicializar Indicador  /*por ejemplo, indicadorPositivo =
                                Verdad */
      Mientras (evaluación de indicador)/*mientras Indicador_
                 Positivo sea Verdad */
           Obtener dato
           Si el dato es negativo
               Actualizar indicador
           En otro caso
               Comprobar Múltiplo de cinco
               Si es múltiplo /*Este si no tiene sino porque no
                              es necesario*/
                   Sumar
               Finsi
           Finsi
      Fin_mientras
      Imprimir resultado de Sumar Múltiplos
FinPP
```

Módulos

Se escribirán sobre el pseudocódigo detallado los comentarios para señalar comienzo y fin de cada uno de los módulos.

Pseudocódigo

```
/*Nombre del programa: Multiplos5*/
     ENTORNO:
     VARIABLES
          Entero numero, suma = 0
```

```
          boolean indicadorPositivo  /*Puede inicializarse aquí
          o antes del mientras, pero ¡ojo! Recuerda que el
          algoritmo debe no hacer nada si la primera vez es
          negativo el número */
```

Programa principal detallado

```
<Inicio>
     indicadorPositivo = Verdad /*Inicializa VCB */
     Mientras (indicadorPositivo == Verdad) /*Compara el valor
                                             del indicador*/
          /*Obtener dato*/
          Escribir ("Escribe un número")
          Leer (numero)
          /*Fin Obtener...*/
          SI (numero < 0)
               indicadorPositivo = Falso /*Cambio de la VCB para
                                          salir de bucle */
          SINO
               SI (numero MOD 5 == 0)
                    suma = suma + numero
               FINSI
          FINSI
     Fin_Mientras
     Escribir ("La suma de múltiplos de 5 es:---",suma)
<Fin_PP>
```

La traza de este pseudocódigo es similar a la del ejercicio anterior, por lo que no se expondrá, sugiriendo al lector que lo haga para comprobar la validez del código.

9. SUBTAREAS ITERATIVAS

En este apartado, se examinarán algunas tareas que se suelen aparecer frecuentemente cuando se diseñan los bucles y que se conocen con el nombre de **subtareas iterativas**.

9.1 Bucles contadores

Son bucles que contienen alguna variable contadora, pero no son bucles controlados por contador, es decir, aunque se usa dentro del bucle el contador no es la variable VCB.

Se dan habitualmente los dos casos siguientes:

1. **Contador de iteración**: su valor termina siendo igual al número de iteraciones que realiza el bucle, sin que ese contador sea la VCB. Se usa cuando se está interesados en evaluar cuántas veces se ejecuta un bucle, sin que sea dicho contador quien controle la ejecución de este bucle. Por

ejemplo, se desea realizar un proceso con números enteros positivos, para lo cuál se utiliza un dato centinela (-1) como final de ejecución del bucle, pero el proceso que interesa en saber cuántos números se han tratado.

2. **Contador de suceso** que cuenta las veces que ocurre un determinado evento, sin que ese contador sea la VCB. Por ejemplo, como en el caso anterior realizar un proceso con números enteros positivos y contar cuántos de ellos son múltiplos de 5. Este contador se incrementará sólo cuando ocurra este suceso.

EJERCICIO PRÁCTICO DE APLICACIÓN

Hacer un algoritmo que permita leer, contar e imprimir caracteres que se leen de teclado, terminando el proceso cuando se lea un punto.

Análisis: En este caso, diseñar un bucle controlado por suceso, siendo el suceso un dato centinela. Punto (.) es el dato centinela.

Entrada: Un carácter.

Salida: El eco del carácter leído y el número total de caracteres procesados.

Suposiciones: La entrada es válida, no hay comprobación de error.

Programa principal generalizado

```
Inicio
  Obtener carácter // lectura anticipada
  Mientras (carácterLeído < > '.') //condición de terminación
       Hacer eco
       Contar caracteres
       Obtener carácter //lectura final
  Fin_mientras
  Escribir Total_caracteres
Fin_PP
```

Pseudocódigo

```
    /*Nombre del programa: ContCar*/
    ENTORNO:
        VARIABLES:
            carácter letra
            entero contletras
```

Programa principal detallado

```
<Inicio>
    contLetras = 0 //Puede hacerse aquí o cuando se declara

    //Obtener carácter
```

```
     Escribir ("Escriba un carácter, . para terminar")
     Leer (letra) /*Letra es la VCB, aquí se inicializa con la
                   lectura anticipada*/
     //Fin Obtener...

     Mientras (Letra<> '.')
         Escribir (Letra)  /*Esto es hacer eco*/

         //Contar caracteres
         contLetra = contLetra+1 /*Este contador no es la VCB*/

         //Obtener carácter
         Escribir("Escriba un carácter, (. para terminar)")
         Leer (letra) /*Actualización, lectura final de la VCB*/
         //Fin Obtener...

      Fin_mientras

     //Escribir total...
     Escribir ("El total de caracteres leídos es ---"
               ,contLetras)
<Fin_PP>
```

¿Se debe inicializar *contLetras*? Sí. Por un lado, si no se inicializa y el primer carácter leído es un '.', saldría en pantalla la basura que contiene la dirección de memoria reservada para *contLetra* en lugar de 0, al imprimir el total de caracteres en el monitor. Por otro lado, si el carácter leído no es el centinela, se entra en el bucle y en este caso, se estaría acumulando 1 a la basura almacenada en dicha dirección de memoria, por tanto, la salida en pantalla también sería incorrecta. ¿Cuántos caracteres se han contado?, uno menos de los caracteres leídos, pues el centinela se ha leído, pero al no entrar en el bucle no se ha contado.

Si se sigue la traza podrá observarse que *contLetras* termina siendo igual a la cantidad de iteraciones que se han realizado.

9.2 Bucles sumadores

Son bucles que contienen alguna variable acumuladora o sumadora. Pueden ser bucles controlados por cualquier tipo de variable, pero la variable sumadora no es la que controla el bucle, es decir, no es la VCB. Se ilustrará el concepto con un ejemplo.

EJERCICIO PRÁCTICO DE APLICACIÓN

Diseñar un algoritmo para leer números enteros y sumar los primeros diez números pares que se introduzcan desde la entrada estándar.

Análisis: Puesto que el proceso que debe repetirse no supone, leer números y sumar no resulta complejo, se prestará especial interés al diseño del bucle, que es el objeto de estudio. El programa pretende leer números enteros cualesquiera por teclado y sumar tan sólo diez que sean pares. Se diseñará un bucle controlado por contador, siendo el contador de números pares la VCB.

```
..........
sumar = 0 // Se inicializa el acumulador
contador = 0 //Se inicializa la VCB
...................
Mientras (contador < 10) /*Compara la VCB*/
    Escribir ("Escriba un número entero")
    Leer (numero)
    Si ((numero MOD 2) = = 0)
        contador = contador + 1/*Varía la VCB, en este caso
                                 incrementando el contador*/
        suma = suma + numero /*Variable acumuladora*/
    Fin_Si
Fin_mientras
.....................
.....................
```

El contador de este ejemplo es un *contador de sucesos*, además de ser la VCB. Este contador no es un contador de iteraciones, se incrementa sólo cuando ocurre el suceso de que el número leído sea par.

Observe el seguimiento de la traza de este trozo de código para verificar que efectivamente realiza la tarea que se requiere.

Mensaje en pantalla	numero	contador	suma
		0	0
Escriba un número entero	15		
Escriba un número entero	4	1	4
Escriba un número entero	-12	2	-8
Escriba un número entero	7		
Escriba un número entero	10	3	2
Escriba un número entero	2	4	4
Escriba un número entero	4	5	8
Escriba un número entero	-2	6	6
Escriba un número entero	-4	7	2
Escriba un número entero	5		
Escriba un número entero	2	8	4
Escriba un número entero	19		
Escriba un número entero	-20	9	-16
Escriba un número entero	27		

Escriba un número entero	32	10	16

10. CÓMO DISEÑAR UN BUCLE

Examinar, evaluar y comprender como funciona un bucle ya construido es una tarea mucho más sencilla que diseñar un bucle que resuelva un determinado problema. Para construir bien un bucle, en primer lugar hay que determinar qué condiciones deben existir al comienzo de cada iteración, para que el bucle funcione correctamente. Al aserto formado por estas condiciones que deben ser verdad al comienzo de cada vuelta, se le llama **invariante del bucle**.

10.1 Invariante del bucle

La condición del *mientras* es parte del invariante del bucle, pero no lo es todo. Las condiciones que también forma parte del invariante son cosas como rango de la variable, inicialización de contadores, actualización de contadores y sumadores, etc.

Se ilustrará el concepto con un ejemplo: Diseñar un algoritmo para sumar los primeros N enteros impares que se lean desde la entrada estándar.

Aunque puede diseñarse un bucle controlado por contador que resolvería el problema, es igualmente acertada la construcción de un bucle controlado por indicador, parar el proceso, que comenzará en falso y cambiará de estado, será verdad, cuando ocurra el suceso "se han leído N números impares".

Asertos que forman parte del invariante:

- El valor del indicador debe comenzar en *falso* pues aún no se han leído *N* números impares. Indica cómo y dónde debe inicializarse la VCB.

- El valor del indicador debe actualizarse a *verdad* cuando se hayan leído *N* números impares. Indica cómo y dónde debe actualizarse la VCB.

- El valor del contador debe empezar en 0 y acabar en *N*, es lo más natural.

- El valor del contador termina siendo igual a la cantidad de números impares que leídos. Esto indica dónde debe actualizarse el contador, justo cuando se lea un número y se haya comprobado que es impar.

- Debe haberse leído un nuevo dato de entrada antes de realizar el proceso, es decir, al comienzo de cada iteración. Esto señala dónde debe realizarse la lectura de datos.

- Debe inicializarse la suma, puesto que se trata de un acumulador. Con esta información se sabe donde debe inicializarse el proceso que va a repetirse.

- Debe actualizarse la suma cuando se haya leído un número impar, incrementándola con el valor de este. Se conoce así el lugar dónde debe actualizarse el proceso que se repite.

El algoritmo debe contener al menos los siguientes pasos:

```
...........
...........
Inicializar indicadorParar a falso /*o indicador_seguir
                                    inicializado a verdad*/
Inicializar contImp a 0
Inicializar la suma a 0
Mientras (indicadorParar == falso)
    Leer (numero)
    Si (numero es impar)
        contImp = contImp+1
        suma = suma +numero
        Si (contImp ==N)
            indicadorParar = Verdad
        Finsi
    Sino
        .......
        .......
    Fin_Si
Fin_Mientras
...................
...................
```

COMO DETERMINAR EL INVARIANTE

Hay un conjunto de preguntas que deben realizarse cuando se analiza un ejercicio iterativo, cuyas respuestas ayudan a concretar el invariante de un bucle favoreciendo así el diseño, haciéndolo eficaz y de calidad. Son las siguientes:

- ¿Cuál es la condición con la que termina el bucle?. Recuerde que la condición de entrada a un bucle es complementaria a la condición de salida. Por ejemplo, si la condición de terminación fuera A < 0, la de entrada al bucle sería A > = 0.

- ¿Cómo debe inicializarse y actualizarse la condición de entrada al bucle?

- ¿Cuál es el proceso que se repite?

- ¿Cómo debe inicializarse y actualizarse el proceso?

Una vez determinadas estas tareas esenciales se puede volver atrás y añadir las demás sentencias que refuerzan el código del programa o realizan trabajos extras. En nuestro anterior ejemplo, se podrían añadir como operaciones extras o líneas que refuerzan el código las que aparecen comentadas a continuación:

```
.....................
Inicializar indicadorParar a falso
Inicializar contImp a 0
Inicializar la suma a 0
```

```
Inicializar contPar a 0  /*Tarea extra para contar pares,
                          contador de suceso*/
Inicializar contTotal a 0 /*Tarea extra para contar todos
                           los números procesados,
                           contador de iteraciones*/
.....................
Mientras (indicadorParar == falso)
    Escribir("Deme un número")  /*Refuerza el código*/
    Leer(número)
    Si (numero es impar)
        contImp=contImp+1
        suma=suma+num
        Si (contImp == N)
            indicadorParar = Verdad
        Finsi
    Sino
        contPar = contPar + 1 /*Tarea extra*/
    Fin_Si
Fin_Mientras
contTotal = contImp + contPar /*Tarea extra*/
Escribir ("Ha escrito --- números, --- pares y ---
          impares", contTotal, contPar, contImp)
          /*Refuerza el código*/
.............................
```

A la vista de las preguntas planteadas, se divide el diseño del bucle en dos actividades:

- Diseño del flujo de control.
- Diseño del proceso que debe repetirse.

A) Diseño del flujo de control de un bucle

A continuación, se exponen las preguntas correspondientes para determinar este propósito:

1. **¿Cuál es la condición de terminación del bucle?**, dicho de otra forma ¿qué debe ser verdad para que el bucle termine?. Una vez determinada la condición de terminación, se puede escribir una parte importante del invariante, la condición de entrada del *mientras*. Generalmente, la condición de terminación se deduce de alguna frase en el enunciado del problema.

   Con la respuesta a esta pregunta se conoce también qué tipo de bucle debe diseñarse, si controlado por contador o controlado por suceso.

Ejemplos:

Frase decisiva en la	Condición de terminación	Condición de entrada

definición del problema		
"Media de las temperaturas de los 365 días del año"	El bucle acabará cuando el contador llegue a 365 (bucle controlado por contador).	El proceso (leer temperatura, realizar cálculos e incrementar el contador) se repite mientras el contador sea menor que 365.
"...hasta que se hayan leído 20 números impares..."	El bucle acaba cuando se hayan leído 20 impares, y el indicador señale este suceso (bucle controlado por suceso, con una tarea iterativa, contador de sucesos).	La iteración se repite mientras el indicador de suceso no cambie su valor, es decir, no se ha llegado a leer 20 impares.
"... el final de la entrada de datos se indica por un '.'..."	El proceso termina cuando se lea el carácter '.' (bucle controlado por centinela).	El bucle se repite mientras el carácter leído sea distinto de '.'

2. **¿Cómo, dónde debe inicializarse y actualizarse la o las VCB's de la condición de entrada?** La respuesta a esta pregunta depende de la condición de terminación, que a su vez depende del tipo del bucle que se haya construido. Es importante considerar que si no se actualiza esta condición, no se saldrá del bucle y se habrá construido un bucle infinito.

   Se responde esta pregunta en razón del tipo de bucle que se diseñe:

   a) Bucle controlado por contador, bucle con contador de iteraciones y el bucle con contador de sucesos.

      Inicialización: los contadores se inicializan normalmente a 0 o 1 antes de la primera iteración del bucle correspondiente. También se puede inicializar al máximo valor, en cuyo caso se decrementará en vez de incrementarse, siempre que se controle bien la cantidad de iteraciones que debe realizar el bucle.

      Actualización:

      - En el bucle controlado por contador y en el bucle con contador de iteraciones, los contadores se actualizan al final de cada iteración.
      - En el bucle con contador de sucesos, el contador se actualiza cada vez que ocurra el suceso en cuestión.

   b) Bucle controlado por centinela.

Inicialización: con la lectura anticipada. Leer antes de la primera iteración.

Actualización: con la lectura final. Lectura dentro del bucle antes de la siguiente iteración, físicamente al final del bucle.

c) Bucle controlado por indicador

Inicialización: el indicador debe tener valor *verdadero* o *falso*, *sí* o *no*, *1* o *0*, fuera del bucle antes de la primera iteración.

Actualización: el valor de la variable indicadora permanece sin variación hasta que alguna condición, dentro del proceso que se realiza en el bucle, provoca el cambio de valor. Por tanto, será este el único caso en que habrá que diseñar el proceso antes de construir la operación de actualización del indicador.

B) Diseño del proceso interior del bucle

Una vez precisada la estructura repetitiva se analizan los detalles del proceso, para lo que se tratará de dar respuesta a las siguientes preguntas:

1. **¿Cuál es el proceso que se va a repetir?** Para estudiar este punto puede ignorarse en principio el bucle, suponiendo de momento que el proceso se va a ejecutar una sola vez. Esto hará que el esfuerzo se centre exclusivamente en la observación de los procesos y no en cuántas veces se ejecutan. Para determinar los procesos, se examinará con detenimiento la definición del problema con todas sus especificaciones.

   Los procesos más frecuentes son: leer, ejecutar cálculos, escribir resultados en pantalla, ordenar listas, etc.

2. **¿Cómo debe inicializarse y actualizarse el proceso que se repite?** Esto dependerá absolutamente de las características del proceso que se realice.

   Por ejemplo, si el proceso fuera *SUMAR*, se inicializará a 0 la variable acumuladora antes de la primera iteración. El proceso se actualizará añadiendo los datos de la suma en cada pasada, antes de acabar la iteración, allí donde la semántica del problema lo requiera.

EJERCICIO PRÁCTICO DE APLICACIÓN

Diseñar un algoritmo para adivinar un número entre 0 y 20 en cinco tiradas.

Análisis: Debería generarse aleatoriamente un número entre 0 y 20, como de momento no se conoce como realizar esta tarea, el número que debe acertar el jugador será almacenado como constante.

EL BUCLE: necesito un bucle que se repita hasta que acierte el número o hasta agotar las tiradas, por tanto, será un bucle controlado por contador, que contará las tiradas, y por indicador, que indicará si se ha acertado o no.

A. Diseño del flujo de control

1. ¿Condición de terminación del bucle?

   Cuando haya acertado (*acertado* vale verdad) *O* cuando haya consumido las tiradas (*contadorTiradas* ha llegado a valer cinco)

   Se puede escribir la condición de entrada que acompañará al *Mientras*:

   Mientras ((**NO** haya acertado *[es decir, acertado vale falso]*) **Y** (**No** haya consumido las tiradas *[aún no ha llegado a valer cinco, contadorTiradas es menor que 5]*))

2. ¿Cómo y dónde debe inicializarse y actualizarse la condición de entrada?

   Inicialización:

   - El contador se inicializa a 0 antes de la primera iteración.
   - El indicador se inicializa a *Falso* antes de la primera iteración.

   Actualización:

   - El Indicador se actualiza a *Verdad* cuando ocurra el suceso "acertar el número secreto".
   - El contador se actualiza incrementándolo en 1 antes del final de cada iteración.

B. Diseño del proceso interior del bucle.

1. ¿Cuál es el proceso que se repite?.
   - Obtener el número
   - Comprobar si ha acertado
   - Contar tiradas
2. ¿Cómo se inicializa y actualiza el proceso?.
   - Obtener el número es una instrucción de lectura que se inicializa dentro del bucle y se actualiza en cada iteración.
   - Los otros dos procesos corresponden a las actualizaciones de las VCB, indicador y contador respectivamente.

Como no existen suposiciones, es responsabilidad del código comprobar que la entrada sea correcta, es decir, un número entre 0 y 20, por tanto, habrá que incluir sentencias que verifiquen este hecho.

Verificar la entrada significa impedir que se introduzcan datos que no pertenecen al rango de valores establecido para dicha entrada, que pueden provocar algún tipo de error en la aplicación o en el sistema. La manera de validar la entrada es construir un bucle controlado por centinela cuyo valor sea precisamente el dato indeseado. El objetivo es impedir que se siga la ejecución del programa hasta que se introduzca un dato válido. Es decir, habrá que construir un bucle del que no se podrá salir a menos que se introduzca un dato correcto. Se estudiará a continuación este nuevo bucle:

BUCLE DE VALIDACIÓN DE ENTRADA: El bucle se repetirá hasta que el número introducido esté entre 0 y 20 incluidos ambos, por tanto, será un bucle controlado por un rango de valores centinelas, siendo *número* la VCB.

A. Diseño del flujo de control

1. ¿Condición de terminación del bucle?

   Cuando ((número leído sea mayor o igual que 0) *Y* (número leído sea menor o igual que 20))

   La condición de entrada de este bucle *Mientras* se escribe, por tanto, del siguiente modo:

   Mientras ((número leído menor que 0) O (número leído sea mayor que 20))

2. ¿Cómo y dónde debe inicializarse y actualizarse la condición de entrada?.

   Inicialización:

   - El número se inicializa por lectura antes de la primera iteración.

   Actualización:

   - El número se actualiza por lectura antes de cada nueva iteración.

B. Diseño del proceso interior del bucle.

   Precisamente, el proceso que se repite es *Obtener el número* y ya se ha comentado como se inicializa y como se actualiza, pues *número* es precisamente la variable que controla el bucle. Por tanto, ya se ha contestado a las preguntas ¿Cuál es el proceso que se repite? y ¿cómo se inicializa y actualiza el proceso?.

Entrada: El número que teclea el usuario.

Salida: El eco del número leído y algunos mensajes en la pantalla referentes al resultado del juego.

Suposiciones: Ninguna.

Programa principal generalizado

```
Inicio
    Inicializar indicador y contador
    Mientras (no haya acertado y no haya consumido las tiradas)
         Obtener y validar número  /*Inicializa y actualiza el
                                   proceso del bucle anterior*/
         Comprobar si ha acertado /*Actualiza el indicador*/
         Contar tiradas /*Actualiza el contador*/
    Fin_mientras
    Presentar Resultados
FinPP
```

Módulos

Se desarrolla el módulo de validación de la entrada:

```
Obtener y validar número
    Obtener el número /*Inicializa la VCB  del bucle siguiente*/
    Mientras ((número leído menor que 0) O (número leído sea
               mayor que 20))
        Obtener el número /*Actualiza la VCB  de este  bucle*/
    Fin_mientras
Fin Obtener y validar número
```

Realice la inspección del pseudocódigo generalizado y del módulo desarrollado y observe que en esta fase del diseño responden a las especificaciones requeridas.

Pseudocódigo

```
/*Nombre del programa: AcertarNum*/
      ENTORNO:
            CONSTANTES
                  numSecreto = 15
            VARIABLES:
                  entero numero, contTiradas
                  boolean acertado
```

Programa principal detallado

```
<Inicio>
      contTiradas = 0
      acertado = Falso
      Mientras (acertado = = Falso Y contTiradas < 5)
            /* Obtener y validar  número*/
            /* Obtener el número*/
            Escribir ("Escribe un n° entre 0 y 20")
            Leer (numero)
            /*Fin Obtener el  número*/
            Mientras ((número leído menor que 0) O (número
                      leído sea mayor que 20))
                  Escribir ("Escribe un n° entre 0 y 20")
                  Leer (numero)
            Fin_mientras
            /*Fin Obtener y validar  número*/
            /*Comprobar si ha acertado*/
            Si (numero = = numSecreto)
                  acertado = Verdad
            FinSi
            /*Fin comprobar*/
            /*Contar tiradas*/
            contTiradas = contTiradas + 1
      FinMientras
      /*Presentar resultados*/
      Escribir ("El número secreto es:", numSecreto)
      Si (acertado = = Verdad)
          Escribir ("Enhorabuena lo ha acertado en --- tiradas",
                    contTiradas)
      SINO
          Escribir ("Ha agotado sus tiradas")
      Finsi
      /*Fin presentar resultados*/
FinPP
```

Se realiza la traza del algoritmo para comprobar si existen errores o se atiene a las especificaciones del enunciado.

Mensaje en pantalla	numero	acertado	contTiradas
		Falso	0
Escribe un n° entre 0 y 20	-5		
Escribe un n° entre 0 y 20	22		
Escribe un n° entre 0 y 20	17		1
Escribe un n° entre 0 y 20	-1		
Escribe un n° entre 0 y 20	0		2
Escribe un n° entre 0 y 20	18		3
Escribe un n° entre 0 y 20	33		
Escribe un n° entre 0 y 20	7		4

Escribe un nº entre 0 y 20	11		5

Ya no vuelve a entrar en el primer *mientras* puesto que se han hecho 5 iteraciones del primer bucle y *contTiradas* ya no es menor que 5. Se pasa a evaluar el *SI* que, como no es cierto, entrará en *SINO* y aparecerá la siguiente salida en pantalla:

```
Ha agotado sus tiradas
```

Observe que se han realizado cuatro iteraciones del bucle de validación, repartidas en tres del primer bucle. Exactamente dos en la primera iteración del primer mientras, una en la tercera iteración del primer mientras y otra en la sexta.

Se propone pasar una nueva traza para forzar el acierto del número y observar el cambio de valor del indicador.

11. ESTRUCTURA DE SELECCIÓN MÚLTIPLE

11.1. Definición y Sintaxis

Una **estructura de selección múltiple** permite presentar varias acciones alternativas y elegir una de ellas en tiempo de ejecución. La selección se realiza por comparación entre una expresión y las constantes correspondientes a las distintas acciones a seguir.

Esta estructura se representa con la sentencia *SEGÚN*.

Su formato o sintaxis es:

```
Según (Expresión) [Hacer]

   para Expresión == constante 1
        Instrucciones A

   para Expresión == constante 2
        Instrucciones B
               ..............

   para Expresión == constante N
        Instrucciones N

   [en otro caso]       //es opcional
        Instrucciones

Fin_Segun
```

A **Expresión** se le conoce como **selector** de la sentencia **Según** y representa una expresión o variable cuyo valor determina la constante seleccionada. Las expresiones pueden ser simples o compuestas.

La lista de constantes (constante 1, 2, etc) contendrá valores del mismo tipo de datos que el selector. Además, cada constante tiene un valor diferente, es decir, no pueden aparecer repetidos valores en la lista. En Java, sólo se permite que tome valores carácter o entero.

¿Cómo se evalúa el proceso? Se evalúa la expresión y se ejecutará el bloque de instrucciones correspondiente a la expresión evaluada como verdadera, pasando a continuación el flujo de control a la línea siguiente a *fin segun*. Si ninguna de las condiciones es verdadera y existe en otro caso, se ejecutará el bloque de instrucciones que contenga.

Se puede observar que realiza el mismo proceso que la estructura *Si-Si no-Si*, pero evaluando las expresiones por comparación con constantes y no con expresiones complejas.

Si se deben ejecutar varios bloques de *Instrucciones* (instrucciones A, B, etc.) en el caso de que sean ciertas varias expresiones, se puede utilizar una expresión compuesta como la siguiente:

```
para expresión = constante 1 o expresión = constante 2
      Instrucciones A
      Instrucciones B
      ..........
```

La sintaxis, en este caso, también dependerá del lenguaje de programación utilizado. Por ejemplo, en Java no se permite que la expresión sea compuesta, pero proporciona herramientas para implementar estos casos.

11.2. Aplicación a procesos controlados por menús

Un menú consiste en presentar en pantalla una ventana con una serie de operaciones, indicaciones u opciones a seguir, cada una de las cuales realiza una función determinada cuya ejecución puede elegir el usuario.

Generalmente, estos menús también contienen por cada opción elegida, otros menús y estos, otros a su vez y, así, sucesivamente. En estos casos, se dice que contiene diversos niveles de menús. Por ejemplo, un cajero automático

Primer nivel de Menú			Segundo nivel de Menú			Tercer nivel de Menú
	saldo			30,00		Marque la cantidad y pulse continuar
	sacar dinero	⇒		50,00		
	movimientos			100,00		

Cuando los menús se presentan anidados, se utilizara la estructura *SEGÚN* anidada y la programación modular, que ayudará enormemente a la comprensión del algoritmo. En estos casos, si pueden aparecer las mismas constantes si están en niveles de anidamiento distintos.

EJERCICIO PRÁCTICO DE APLICACIÓN

Diseñar un algoritmo que lea dos números de teclado y calcule la suma, resta, multiplicación y división, según la opción deseada y permita, además, repetir el proceso mientras lo desee el usuario.

Análisis: Como especifica el enunciado, el algoritmo debe ser iterativo, por tanto, es conveniente diseñar un menú que presente en pantalla todas las posibles opciones, incluida una que permita al usuario la posibilidad de terminar el proceso. Si en el transcurso de resolución del algoritmo surge la conveniencia de construir nuevos bucles, debe igualmente dedicar el esfuerzo necesario para diseñarlos correctamente.

EL BUCLE: Se construirá un bucle que se repita hasta que el usuario decida pulsar la tecla prefijada para terminar el proceso, por tanto, será un bucle controlado por centinela, por ejemplo, puede ser *F*, de finalizar, el valor del centinela.

A. Diseño del flujo de control

1. ¿Condición de terminación del bucle?: Sea *Opcion* la variable de control de bucle. Cuando quiera finalizar, se puede modelar como *opcion = 'F'*. La condición de entrada al *Mientras* puede expresarse como: mientras no se desee finalizar, es decir

```
Mientras (Opcion <> 'F')
```

2. ¿Cómo y dónde debe inicializarse y actualizarse la condición de entrada?

   Inicialización: como se trata de un centinela, mediante lectura anticipada:

   - Presentar el *MENU* para poder elegir la opción conveniente.
   - Elegir la opción.

   Actualización: mediante lectura antes de la siguiente iteración o lectura final:

   - Presentar el *MENU* para poder elegir de nuevo seguir o finalizar
   - Elegir la opción deseada

B. Diseño del proceso interior del bucle.

1. ¿Cuál es el proceso que se repite?.
   - Obtener los números
   - Realizar la operación (Según la opción elegida)
   - Presentar *MENU* forma parte de la actualización de la VCB.
   - Elegir la opción forma parte de la actualización de la VCB.
2. ¿Cómo se inicializa y actualiza el proceso?.
   - Obtener los números se actualiza dentro del bucle en cada iteración con la lectura de los números.
   - Realizar la operación se actualiza con los cálculos realizados en cada vuelta.
   - Los otros dos procesos corresponden a la visualización en pantalla del menú de opciones y la actualización por lectura de la VCB.

Entrada: La opción elegida y los números con los que se desea operar.

Salida: Eco de los datos de entrada y resultados de las operaciones con los mensajes correspondientes.

Suposiciones: Ninguna.

No se ha realizado ninguna suposición, por tanto, hay que estudiar qué tratamiento se realizará para impedir errores en la entrada de datos. Cuando se elija la opción deseada, se incluirá un procedimiento para impedir que se escriba una opción incorrecta, es decir, no incluida en el rango de valores válidos para la entrada.

Se llamará a este módulo *Elegir y validar Opción*. No se detallará porque su estructura es similar a la del último ejercicio realizado.

Programa principal generalizado

```
Inicio
      Presentar MENU
      Elegir y validar Opción  /*Inicializa la variable de
      control del bucle */
      Mientras (opcion <> 'F')  //mientras no quiera finalizar
            Obtener números   /*Inicializa y actualiza el
                               proceso*/
            Según (Opcion)
                  Para opcion = = 'S'
                        Realizar SUMAR
```

```
                  Para opcion = = 'R'
                        Realizar RESTAR
                  Para opcion = = 'M'
                        Realizar MULTIPLICAR
                  Para opcion = = 'D'
                        Validar Numero2 y DIVIDIR
            Fin Según
            Presentar resultados
            Presentar MENU
            Elegir y validar Opcion  /*Actualiza la variable de
                                      control del bucle */
      Fin_mientras
FinPP
```

En la construcción del módulo **Elegir y validar Opción** se ha impedido que se introduzca una opción incorrecta, es decir, sólo pueden introducirse los valores permitidos, que son los que aparecen en pantalla en el módulo **Presentar MENU**. Observe que en este caso existen varios datos centinelas, tantos como valores válidos de la opción. Sin embargo, el número leído, que es otro dato de entrada, no se ha validado, tan sólo se ha impedido que se realice una operación inválida, dividir entre 0. El resto de las operaciones si puede realizarse sin problemas.

En este nivel puede observarse que el algoritmo responde a los requisitos establecidos en la fase de análisis, por tanto, puede darse por bueno el diseño y pasar al siguiente nivel de detalle.

Pseudocódigo

```
/*Nombre del programa: MenuOperaciones*/
      ENTORNO:
            CONSTANTES
                  No hay.
            VARIABLES:
                  entero numero1, numero2
                  real resultado
                  carácter opcion
```

Programa principal detallado

```
Inicio
     /* Presentar MENU*/
     Escribir ("Pulse S para SUMAR")
     Escribir (" R para RESTAR")
     Escribir (" M para MULTIPLICAR")
     Escribir (" D para DIVIDIR")
     Escribir (" F para FINALIZAR")
     Escribir("Pulse la tecla correspondiente a la operación que
               desee realizar")
     /*Fin Presentar MENU*/
```

```
     /* Elegir y validar Opción */
     leer (opcion)
     Opcion = Mayúscula (opcion)  /*De esta manera convertimos a
     mayúscula la tecla pulsada para evitar tener que preguntar
     a continuación la posibilidad de haber pulsado minúsculas*/

     Mientras (opcion != 'S' y opcion != 'R' y opcion != 'M' y
               opcion != 'D' y opcion != 'F')
          <Presentar MENU>        /*Aquí se debe escribir el
          trozo de código correspondiente al módulo Presentar
          MENU descrito anteriormente.*/
          Leer (opcion)
          Opcion = Mayúscula (opcion)
     Fin Mientras
     /*Fin elegir y validar Opción*/
     Mientras (opcion != 'F')
          /*Comienza Obtener números*/
          Escribir ("Escriba dos números enteros")
          Leer (numero1, numero2)
          /*Fin Obtener números*/
          Según (Opcion)
               Para opcion = = 'S'
                    resultado = numero1 + numero2 /*Realizar
                                                   SUMAR*/
               Para opcion = = 'R'
                    resultado = numero1 - numero2 /*Realizar
                                                   RESTAR*/
               Para opcion = = 'M'
                    resultado = numero1 * numero2 /*Realizar
                                                   MULTIPLICAR*/
               Para opcion = = 'D'
                    /*Llamar a Validar Numero2 y DIVIDIR*/
                    Mientras (numero2 = = 0)
                         Escribir ("No se puede dividir entre 0,
                                    escriba otro número")
                         Leer (numero2)
                    Fin Mientras
                    resultado = numero1 / numero2
          Fin Según
          /*Presentar resultados*/
          Escribir ("El resultado de la operación es:",
                     resultado)
          <Presentar MENU> /*Aquí debe colocarse el código de
                            este módulo */
          <Elegir y validar Opcion>  /*debe colocarse el código
                                       de este módulo */
     Fin_mientras
FinPP
```

Se han escrito en el pseudocódigo del programa principal detallado los nombres de los módulos **Presentar MENU** y **Elegir y validar Opción** con el único propósito de no volver a repetir el grupo de sentencias correspondiente. En pseudocódigo, puede permitirse esta pequeña licencia con el objetivo de hacerlos más cortos y más fáciles de leer, pero no puede hacerse así cuando se utilice el lenguaje de compilación para codificar el algoritmo.

12. OTRAS ESTRUCTURAS REPETITIVAS

12.1 Bucles Repetir

La sentencia **Repetir** es una estructura de control de flujo iterativa en la que la expresión del bucle se evalúa al final del mismo. Este formato garantiza que el bucle se ejecutará al menos una vez.

Existen dos maneras de construir este tipo de bucle, cuyos formatos son:

1.
```
Repetir
      Instrucciones
Mientras (Expresión)
```

2.
```
Repetir
      Instrucciones
Hasta (Expresión)
```

Como en el caso de las anteriores estructuras, las *Instrucciones* pueden ser una o varias y la *expresión* es una condición booleana que puede ser simple o compuesta.

El proceso se evalúa del modo siguiente: se ejecutan todas las *Instrucciones* existentes entre *Repetir* y *Mientras*, a continuación se evalúa la *expresión*, si es cierta se repite el proceso, en caso contrario el flujo de control continúa por la línea siguiente al *Mientras*. En el caso del bucle *Repetir-Hasta*, el proceso se repite mientras la expresión sea falsa y terminará cuando la expresión sea verdadera, es decir, se repite hasta que se cumpla la expresión.

Flujo de control.- los diagramas correspondientes al flujo de control son los siguientes:

Primer formato Segundo formato

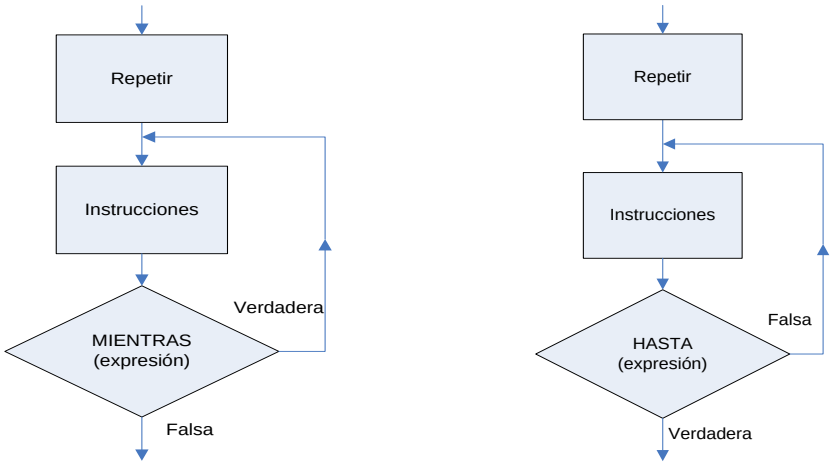


Figura 3.3. Flujo de control en las estructuras repetitivas Repetir

Observe que la sentencia *Repetir...Mientras* utiliza la expresión para decidir cuándo *entrar* en el bucle, a diferencia de la sentencia *Repetir...Hasta*, que utiliza la expresión para decidir cuándo parar la iteración.

Para trasformar un bucle *Repetir-Mientras* en un bucle equivalente que utiliza la sentencia *Repetir-Hasta*, las expresiones deben complementarse. Recuerde que cuando una expresión es verdadera, su complemento será falso, y viceversa.

Todo lo estudiado en cuanto al diseño del flujo de control para los bucles *Mientras* también es aplicable para este nuevo tipo de bucle en sus dos formatos. La VCB se evalúa con la *Expresión* que acompaña a *Mientras*, se inicializa antes de la primera iteración y se actualiza antes de cada nueva iteración. La diferencia está en el orden físico de las sentencias. Físicamente, la inicialización y la actualización coinciden en los bucles *Repetir* y están dispuestas en lugares diferentes en el bucle *Mientras*.

Los bucles *Repetir* son especialmente útiles para la validación de datos de entrada, puesto que al menos se está interesado en leerlos una vez y este bucle garantiza que el proceso se realizará de esta forma.

Por ejemplo, supongamos que queremos diseñar un programa para procesar de forma iterativa calificaciones de exámenes que se leerán desde la entrada estándar. Sabemos que la puntuación debe estar en el rango de valores de 1 a 10, ambos inclusive. Para validar esta entrada se realizará un módulo que asegure que la calificación leída esté dentro del anterior intervalo.

Puede plantearse cualquiera de las dos soluciones siguientes, evaluar ventajas e inconvenientes y decidir cuál es la que más interesa para el caso que se desea modelar. Esto es posible si el lenguaje permite la implementación de ambas soluciones.

Solución Repetir...Mientras

```
.......
Repetir
   Escribir ("Introduzca una nota (rango: [1,10])")
   Leer (nota)
Mientras (nota <1 o nota >10)
```

Solución Repetir...Hasta

```
.......
Repetir
   Escribir ("Introduzca una nota (rango: [1,10])")
   Leer (nota)
Hasta (nota >=1 y nota <=10)
```

Como se utilizará JAVA para codificar los algoritmos, de los dos formatos anteriores se usará siempre el *Repetir-Mientras*, que es el que incorpora este lenguaje de programación.

A continuación, se realiza el mismo ejemplo con un bucle **Mientras,** para comparar su ejecución con las dos soluciones anteriores. Recuerde que la solución **Mientras** requiere una lectura anticipada de la VCB y una lectura final.

Solución Mientras

```
.......
Escribir ("Introduzca una nota (rango: [1,10])")
Leer (nota)
Mientras (nota <1 o nota >10)
   Escribir ("Introduzca una nota (rango: [1,10])")
   Leer (nota)
Fin Mientras
```

La sentencia *Mientras* se conoce con el nombre de **bucle pretest**, precisamente por realizar la comparación antes de ejecutar el cuerpo del bucle, mientras que la sentencia *Repetir* lo hace al final y es conocida como **bucle postest**.

12.2. Bucle controlado por contador PARA

La sentencia *Para* no es una estructura estándar, pero viene implementada en la mayoría de los lenguajes de programación. Este tipo de bucle no es de propósito general, se ha diseñado pensando expresamente en los bucles controlados por contador, para simplificar su escritura y mejorar su velocidad de ejecución.

Su formato es el siguiente:

```
Para (Expresión1, Mientras Expresión2, Expresión3)
      Instrucciones
Fin_Para
```

Como en las estructuras anteriores, las I*nstruccione*s pueden ser una o varias, y las expresiones pueden ser simples o compuestas.

Expresión1 se corresponde con la inicialización de la VCB, el contador.

Expresión2 representa la condición con la que se evalúa el contador.

Expresión3 corresponde a la actualización de la VCB, es decir, el incremento o decremento del contador, que no necesariamente tiene que ser de uno en uno.

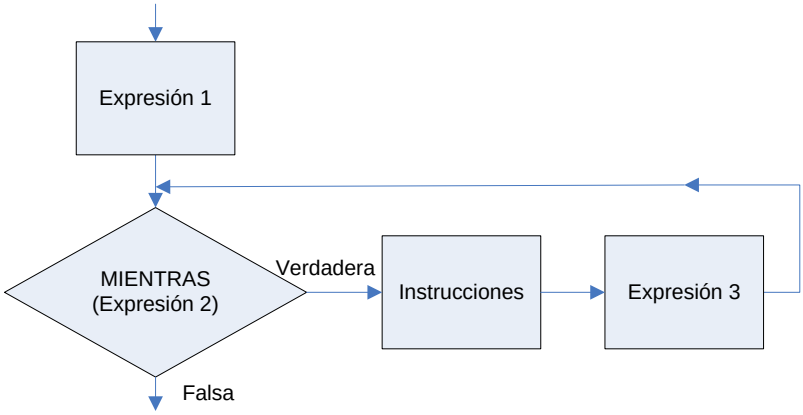


Figura 3.4. Flujo de control en los bucles Para

El proceso se realiza del siguiente modo: se inicializa el contador según *Expresión1*, a continuación se evalúa la VCB, según *Expresión2*, si la condición es cierta, se ejecutará el cuerpo del bucle, representado por *Instrucciones*, posteriormente se incrementa o decrementa la VCB conforme a *Expresión3*. Se repite el proceso, volviendo a evaluar la condición para la iteración del bucle. Cuando esta condición sea falsa, se sale del bucle y sigue el flujo de control por la línea siguiente a *fin para*.

Para hacer un uso racional e inteligente de los bucles *Para* deben tenerse en cuenta los siguientes aspectos:

- La variable de control del bucle (VCB) no debe actualizarse desde dentro del bucle. Su valor puede utilizarse dentro, pero no variarlo, salvo que así lo requiera la semántica del ejercicio, en cuyo caso no deberá aparecer *expresión3*. Es decir, la VCB puede aparecer en cualquier expresión pero no en la parte izquierda de una sentencia de asignación.

- La VCB se incrementa o decrementa en el valor que aparezca en *Expresión3*, que no necesariamente debe ser 1.

- *Expresión2* puede ser una condición compuesta y contener no sólo la condición que evalúa el contador, sino cualquier otra condición de terminación adicional. Es decir, puede contener la VCB de un bucle controlado por contador y la condición de un bucle controlado por suceso si es semánticamente correcto. Por ejemplo, el ejercicio de realizar un programa para acertar un número entre 0 y 20 en cinco tiradas puede

diseñarse con un bucle Para donde las VCB's son un contador y un indicador.

13. CRITERIOS PARA LA ELECCIÓN DE UNA SENTENCIA ITERATIVA

Puesto que se dispone de varias estructuras iterativas para la construcción de bucles, es bueno comentar algunos criterios que ayudarán a elegir la sentencia más adecuada, según el contexto de nuestro algoritmo.

- Si se trata de un bucle controlado por contador, utilice siempre una sentencia *Para*. Si es un bucle controlado por contador y suceso, use *Para* o *Mientras*. Pero si el número de veces que ha de repetirse el bucle es grande, es preferible utilizar *Para*, pues el compilador lo evalúa y procesa más rápidamente.

- Si es un bucle controlado por suceso y el cuerpo del bucle debe realizarse al menos una vez, lo más apropiado es utilizar Repetir.

- Si se trata de un bucle controlado por suceso y no se sabe nada acerca de la primera ejecución, es conveniente usar *Mientras*.

- Si son apropiados los dos bucles, *Mientras* y *Repetir*, se aconseja usar el que mejor refleje el contexto del algoritmo o se adapte a nuestras preferencias.

- Por último, siempre que exista alguna duda usar *Mientras*.

14. NOTAS SOBRE EL USO DE NÚMEROS ALEATORIOS

La mayoría de los lenguajes de programación tienen implementadas las funcionalidades necesarias para hacer uso de los números generados al azar. Cuando en un programa se necesita manejar números aleatorios, bastará con usar la función incorporada por el lenguaje para crearlos. La forma de generarlos depende de cada lenguaje de programación en particular. La forma de generarlos depende de cada lenguaje en particular. En Java se puede usar la clase **Random** de **java.util**, o la función estática **random** de la clase **Math**.

En pseudocódigo se seguirá la siguiente sintaxis:

```
numeroGenerado =  GNA ()
```

Donde **GNA** (Generar Número Aleatorio), representa una función que genera números reales mayores que 0.0 y menores que 1.0, nunca 1 y cuyo valor es almacenado en la variable `numeroGenerado`.

Si por ejemplo, necesitamos generar un número entre 0 y 100 no incluído, basta con multiplicar por 100 y truncar este número, es decir, quedarse la parte entera del

número resultante. Podemos asegurar que en `numeroGenerado` se almacena un número mayor o igual que 0 y menor que 100, que aparecerá en pantalla tras la ejecución de la sentencia *Escribir*.

Si el número generado por GNA es por ejemplo 0,0012, al multiplicar por 100 y truncar `numeroGenerado` sería 0. Si el número generado por GNA fuera 0,9999 `numeroGenerado` sería 99.

La idea es similar a sacar una bolita correspondiente a un número del bombo que contiene todos los números válidos en el juego de la BONOLOTO o el BINGO o cualquier otro juego de azar.

EJERCICIO PRÁCTICO DE APLICACIÓN

Se desea visualizar en pantalla un número generado al azar entre 0 y un número leído por teclado, este último incluído.

El ejercicio servirá tan sólo para ilustrar el uso de números aleatorios, por tanto, y al ser demasiado simple no resulta necesario realizar el análisis.

Pseudocódigo

```
/*Nombre del programa: Aleatorio*/
     ENTORNO:
          VARIABLES:
               entero: numero, numeroGenerado
```

Programa principal detallado

```
<Inicio>
     Escribir ("Escribe un número")
     Leer (numero)
     numeroGenerado = (parte entera)(GNA()*(numero+1))
     Escribir ("El número generado es ---", numeroGenerado)
Fin_PP
```

Si el número leído es 20 multiplicaremos el número generado con GNA por 21 y truncamos. Por ejemplo, si el número generado por GNA es 0,0001 al multiplicar por 21 y truncar `numeroGenerado` almacenaría un 0. Si el generado por GNA fuera 0,9999 tendríamos `numeroGenerado` igual a 20.