CAPITULO 4. BUCLES

Ángel Fidalgo Blanco
Universidad Politécnica de Madrid

UNIDAD DIDACTICA Nº 4.
BUCLES.

OBJETIVOS:

* Conocer el concepto de bucles.
* Identificar las distintas partes de un bucle.
* Conocer los distintos tipos de bucles.
* Representar mediante algoritmos los distintos tipos de bucles.

CONTENIDOS:

4.1. Introducción.

4.2. Tipos de bucles.

4.2.1. Bucles For.
4.2.1.1. Concepto del bucle For.
4.2.1.2. Representación algorítmica del bucle For.

4.2.2. Bucles While.
4.2.2.1. Concepto del bucle While.
4.2.2.2. Representación algorítmica del bucle While.

4.2.3. Bucles Do-While.
4.2.3.1. Concepto del bucle Do-While.
4.2.3.2. Representación algorítmica del bucle Do-While.

CONOCIMIENTOS PREVIOS:

* Asignaciones de variables.
* Expresiones.
* Sentencias de entrada y salida.
* Sentencia if-else.

4.1. INTRODUCCIÓN.

En programación se denomina bucle a la ejecución repetidas veces de un mismo conjunto de sentencias. Normalmente en cada nueva ejecución varía algún elemento. Para comprender mejor el concepto de bucle imaginemos una máquina de tren que parte de la estación "inicio" y viaja hasta la estación "final"; cuando el maquinista llega a la estación "final" vuelve a la estación "inicio" y así sucesivamente. Cada vuelta que da el maquinista es una ejecución o una iteración, en la cual se ejecutan todas las instrucciones que hay en el trayecto comprendido entre la estación "inicio" y la estación "final". La **figura nº 1** muestra este concepto.

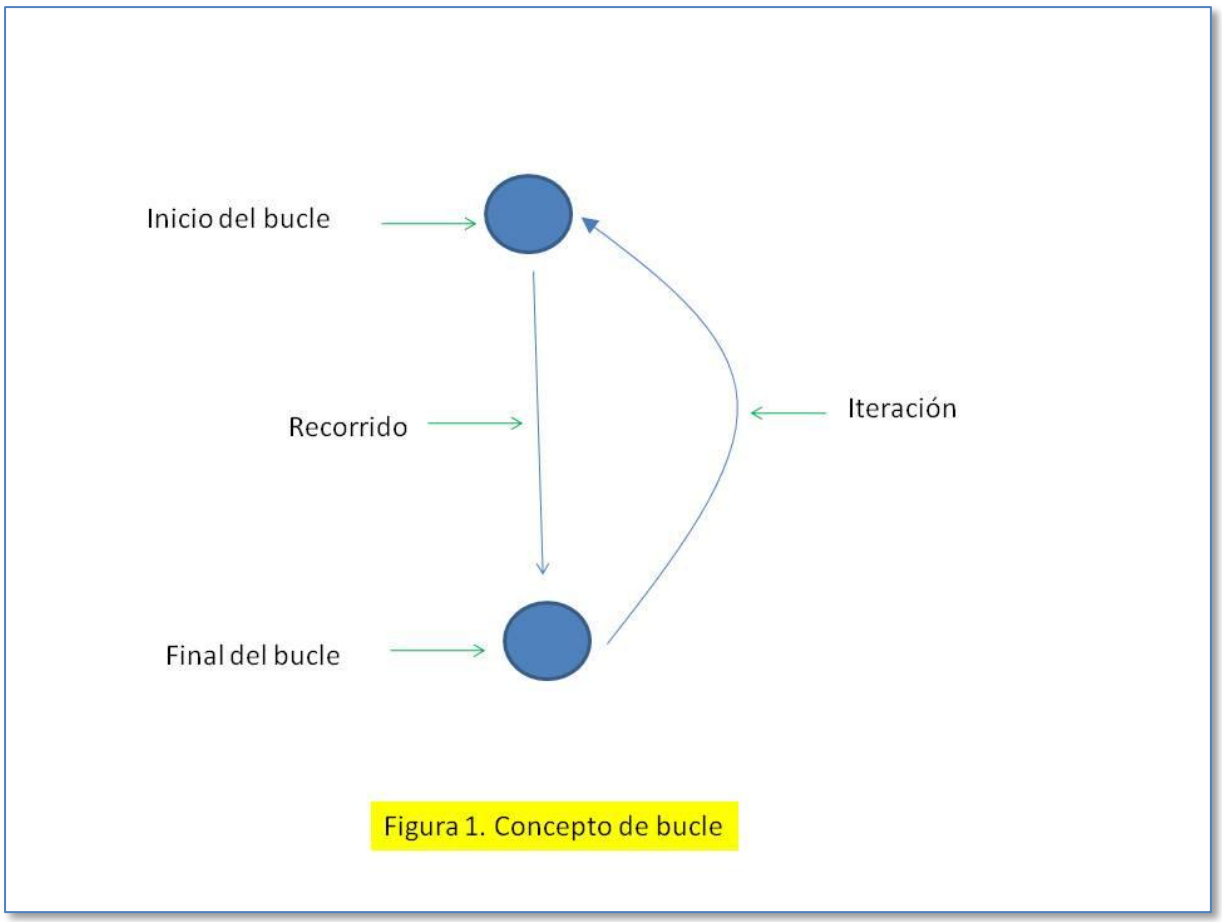

Figura 1. Concepto de bucle

Para realizar un bucle correctamente, el maquinista tiene que conocer tres cosas: el comienzo, el final del bucle y el número de iteraciones que tiene que realizar. Para nuestro maquinista es fácil reconocer el inicio y el final del bucle; pero además alguien tiene que decirle el número de vueltas que tiene

que realizar, de otra forma estaría dando vueltas indefinidamente. **La forma de indicar al "maquinista" el número de vueltas define el tipo de bucle**. Así pues, cuando utilicemos las instrucciones de bucles debemos indicar al programa donde empieza el bucle, donde termina y cuantas iteraciones tiene que realizar. Todas las sentencias comprendidas entre el comienzo y el final del bucle se ejecutarán en cada iteración.

Independientemente de qué tipo de bucle estemos utilizando se debe indicar el inicio, el final y el número de iteraciones; aunque para cada tipo de bucle se especifica de una forma distinta. Esto también es aplicable para los distintos lenguajes de programación, cada lenguaje define un bucle de forma distinta; pero la utilización y los componentes son los mismos para todos los lenguajes. A continuación se analizarán los distintos tipos de bucles con su representación algorítmica.

4.2. TIPOS DE BUCLES.

Existen, principalmente, tres tipos de bucles, su utilización depende del tipo de programa que estemos realizando. Los bucles se definen básicamente por la forma en que les indicamos el número de iteraciones que debe realizar.

4.2.1. Bucles For.

4.2.1.1. Concepto de bucle For.

Los bucles **for** se utilizan cuando el programa sabe el número de vueltas que tiene que realizar cuando entra en el bucle. El número de vueltas se puede indicar por una constante (por ejemplo "5" vueltas) o por una expresión (por ejemplo "n" vueltas o "n*j" vueltas), en este último caso cuando el maquinista llega al inicio del bucle la variable debe estar asignada o la expresión calculada.

Para llevar la cuenta del número de iteraciones que se realizan en el bucle se utiliza una variable entera, de esta forma, en todo momento se sabe el número de iteración sin más que observar el valor de la variable, esta variable también se utiliza para saber cuándo se debe acabar el bucle.

En este tipo de bucles se debe especificar al comienzo del mismo la variable entera que se utilizará para contar el número de iteraciones y el número de vueltas que debe realizar el bucle. Para evitar posibles errores en el valor de la variable, al comienzo del bucle se debe especificar el valor inicial y el valor final que tendrá la variable entera, de esta forma se tendrá controlado en todo momento el número de iteraciones. También es conveniente especificar el incremento de la variable cada vez que se realiza una iteración (el 97% de las veces el incremento es 1).

Por ejemplo si deseamos utilizar la variable entera **I** para controlar el número de iteraciones y queremos que realice 4 iteraciones, entonces se podría especificar de la siguiente manera:

I = 1,4,1

Esto significa que al comenzar el bucle la variable toma el valor 1 y finaliza el bucle cuando toma el valor 4. También se podría especificar que el bucle realizara 4 iteraciones de otras formas:

I = 2,5,1
I = 28,31,1
I = n,n+3,1
I = 2,8,2
I = 3,10,2

Si generalizamos, se podría representar el inicio del bucle como **I = expr1, expr2,inc** donde expr1 y expr2 representan expresiones enteras e inc el incremento que toma la variable I. La condición para que el bucle realizara 4 iteraciones sería que la parte entera de **(expr2 +inc - expr1)/ inc = 4**. Compruebe el lector que esta condición se cumple para todos los casos citados anteriormente. La fórmula genérica para calcular el número de iteraciones que se realizará dentro de un bucle **for** es:

Parte entera ((expr2 + inc - expr1)/ inc)

Obsérvese que cuando comienza el bucle se hace cumplir la condición de que **I=expr1 (cond1)** y cuando finaliza el bucle es porque no se cumple la condición **I<=expr2 (cond2)**, el incremento se expresa por otra expresión **(cond3)** (por ejemplo incremento 1 sería I=I+1).

Cada vez que un bucle **for** realiza una iteración el valor de la variable I cambia. Siempre que comience el bucle el valor de la variable I se hace igual al valor de la primera expresión, es decir, **I=expr1**. Posteriormente, el valor de I será igual al valor que tenía antes mas el incremento; es decir **I=I+inc**. Veamos dos ejemplos:

Ejemplo 1. Los valores que toma la variable I en el bucle I=2,5,1 son los siguientes:

Primera iteración I=2
Segunda iteración I=3
Tercera iteración I=4
Cuarta iteración I=5

Ejemplo 2. Para el caso de I=3,10,2 serían:

Primera iteración I=3
Segunda iteración I=5
Tercera iteración I=7
Cuarta iteración I=9

La finalización del bucle se detecta por el valor que toma la variable I, así en el primer ejemplo en la última iteración el valor de I es igual a 5, el bucle intenta realizar una nueva iteración y el valor de I toma el valor 6 (5+inc) y este valor supera al límite (6>5) por tanto no se realiza el bucle. En el segundo caso, el bucle intenta realizar una quinta iteración y el valor de I se hace igual a 11 (9+inc), este valor supera al límite (11>10) por tanto el bucle no se realiza.

Cuando el bucle finaliza, el programa continúa la ejecución en la primera instrucción que encuentre después del final del bucle.

4.2.1.2. Representación algorítmica del bucle for.

En algoritmia el comienzo del bucle for siempre se representa por una especie de hexágono "estirado" (ver la **figura nº 2**). Dentro del hexágono se debe especificar el comienzo y el final del bucle; es decir, la condición inicial y la condición final. El final del bucle se especifica con un punto negro. En la figura se representa el algoritmo para un bucle **for** con 4 iteraciones.

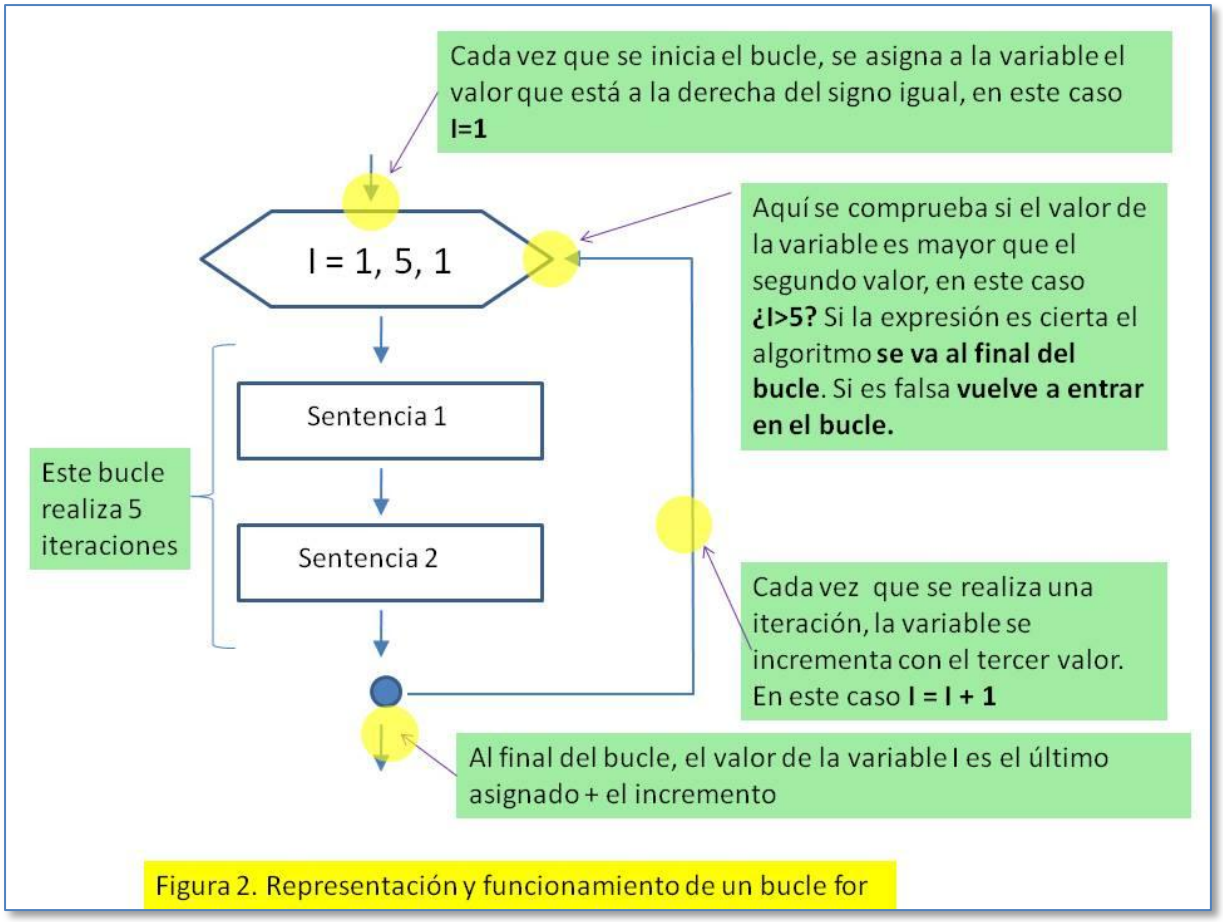

Figura 2. Representación y funcionamiento de un bucle for

El valor de la variable entera que se utiliza como contador sufre un proceso de cambio dentro del bucle, cada vez que ocurre una iteración se suma el valor especificado en la expresión de incremento. Pero, ¿qué ocurre con el valor de la variable anteriormente citada antes y después de que se ejecute el bucle?. La mayoría de mis alumnos han tenido muchos problemas en la realización de los programas por no considerar la anterior pregunta, por tanto conviene tener en cuenta qué ocurre con la variable que se utiliza como contador.

Antes del bucle la variable entera puede tener cualquier valor, es decir si la hemos asignado un valor determinado, será este valor el que contenga; pero atención, **el valor de la variable cuando comienza el bucle se sustituye por el valor indicado en la expr1**, de esta forma el valor anterior se destruye y se carga uno nuevo. Así pues se debe prestar especial atención a esta variable.

Después del bucle la variable entera contiene el valor de la última iteración

más el incremento. Este es un factor de vital importancia puesto que se suele creer que el valor de la variable tiene el último valor asignado en la última iteración. La **figura nº 3** representa un algoritmo que muestra los valores de la variable "contador" antes y después de ejecutar un bucle **for**. No existe ningún problema en utilizar esta variable después del bucle teniendo en cuenta las consideraciones anteriores.

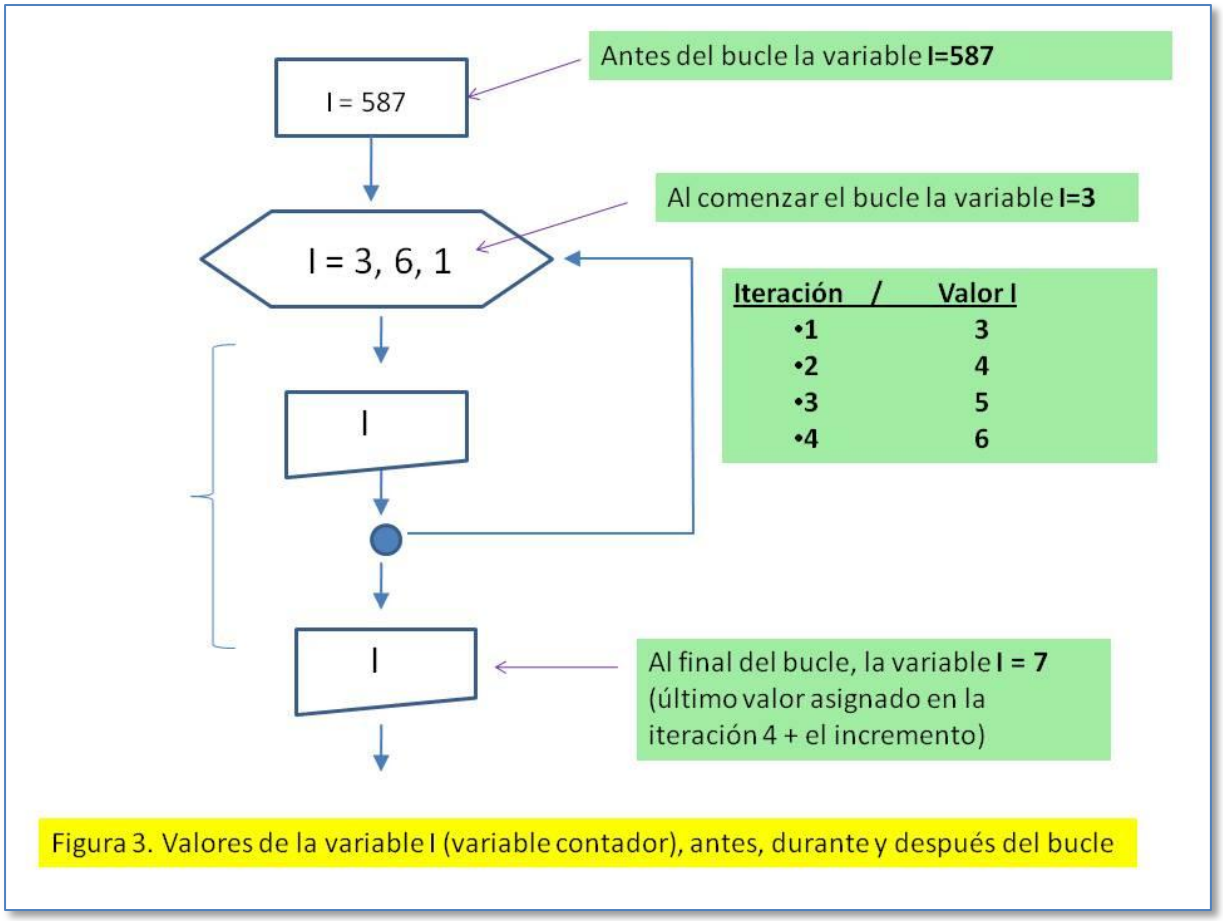

Figura 3. Valores de la variable I (variable contador), antes, durante y después del bucle

Por norma general las variables que utilicemos como contadores en los bucles **for** no las utilizaremos a lo largo de todo el programa. **No se debe cambiar el valor de la variable dentro del bucle for** ya que esto "despistaría" al bucle y causaría un mal funcionamiento del mismo.

Se suele utilizar la variable del contador del bucle **for** dentro del mismo para realizar ciertas operaciones, ya que ésta suministra una secuencia de valores que se pueden utilizar para realizar ciertas operaciones. Las **siguientes figuras 4, 5, 6 y 7** diversas utilizaciones de la variable de

contador dentro de un bucle.

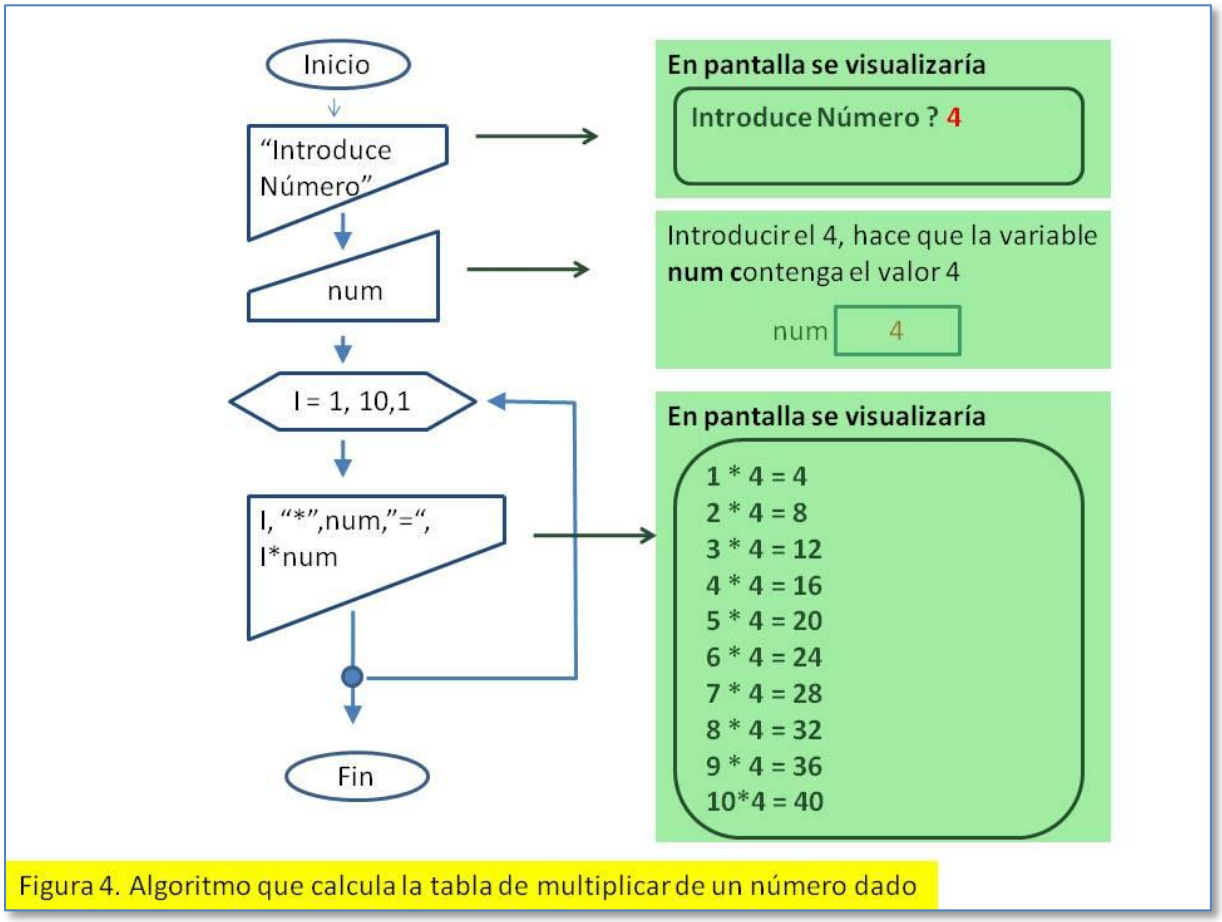

Figura 4. Algoritmo que calcula la tabla de multiplicar de un número dado

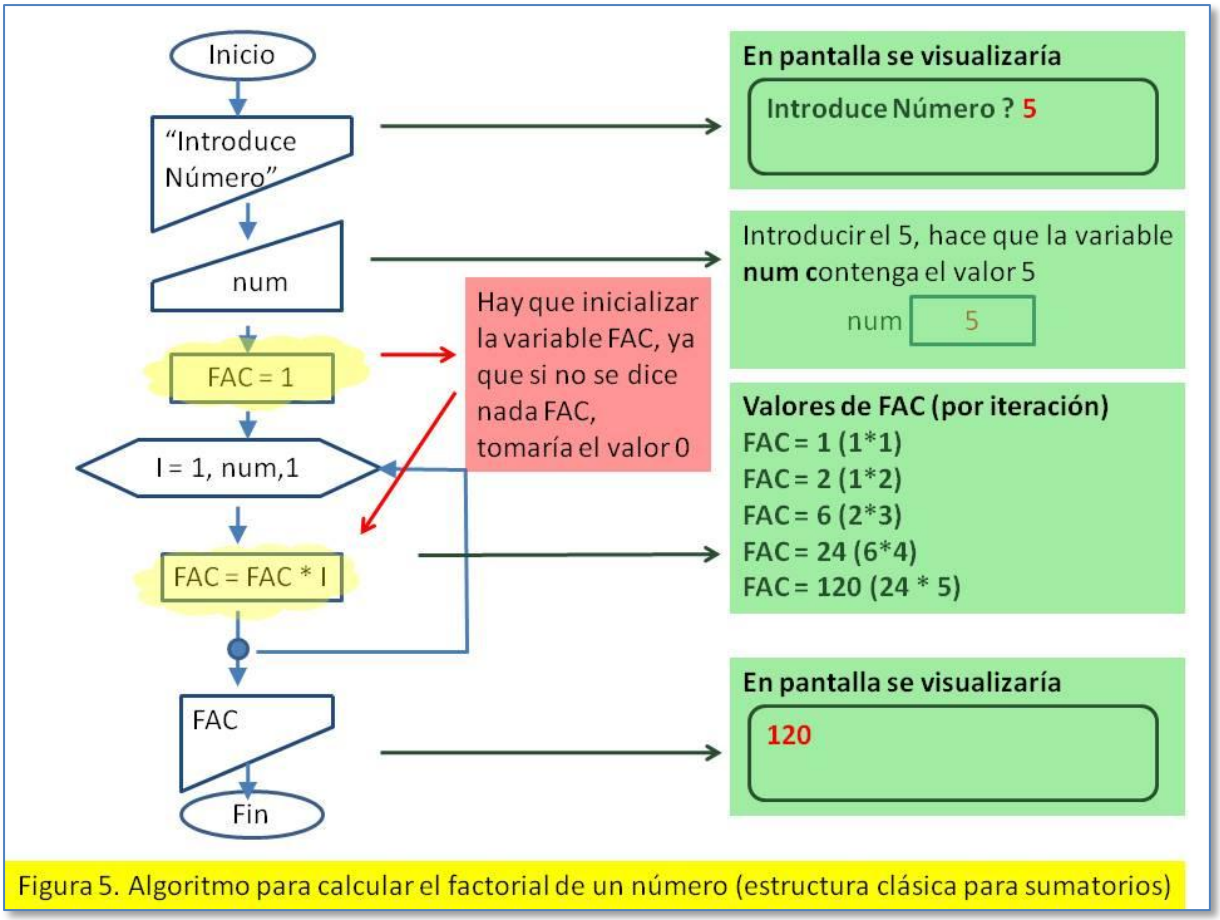

Figura 5. Algoritmo para calcular el factorial de un número (estructura clásica para sumatorios)

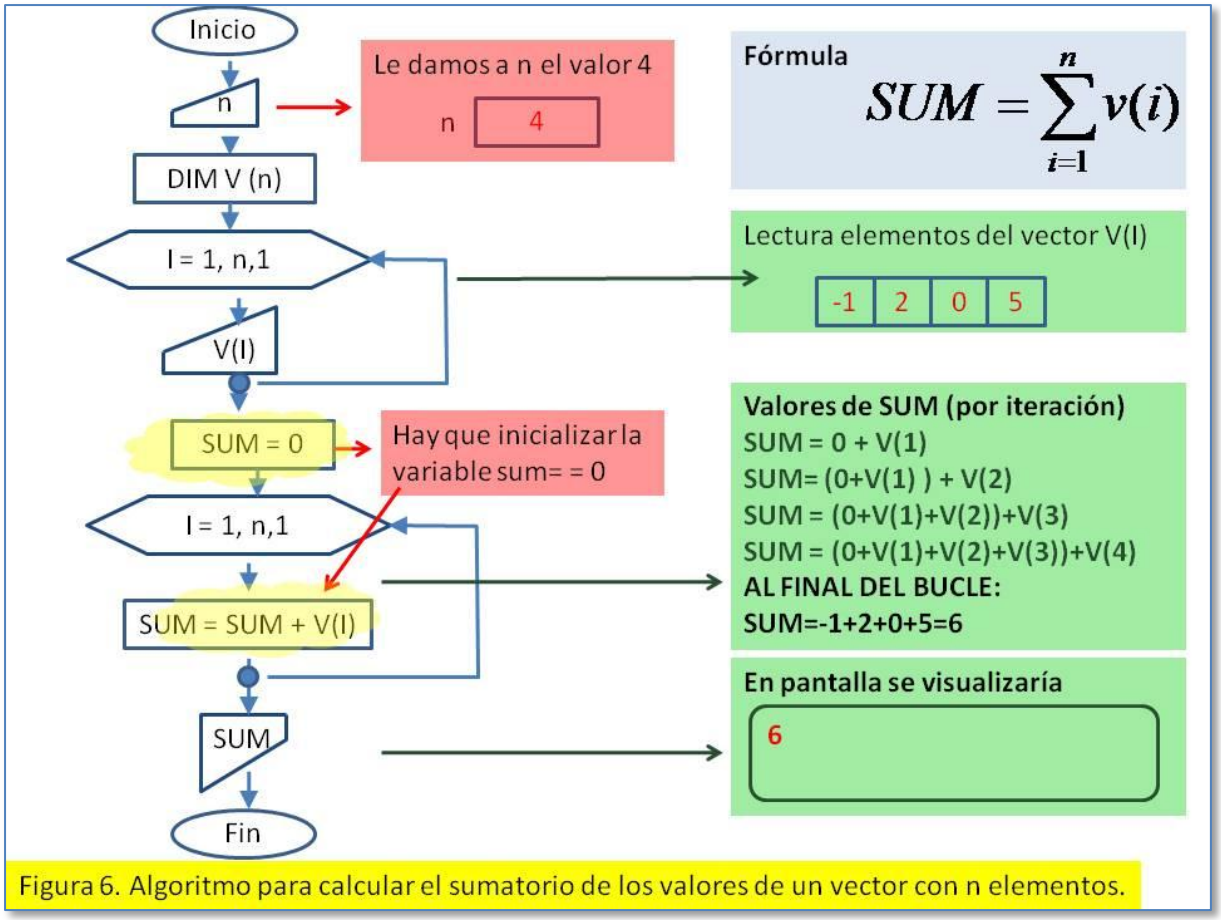

Figura 6. Algoritmo para calcular el sumatorio de los valores de un vector con n elementos.

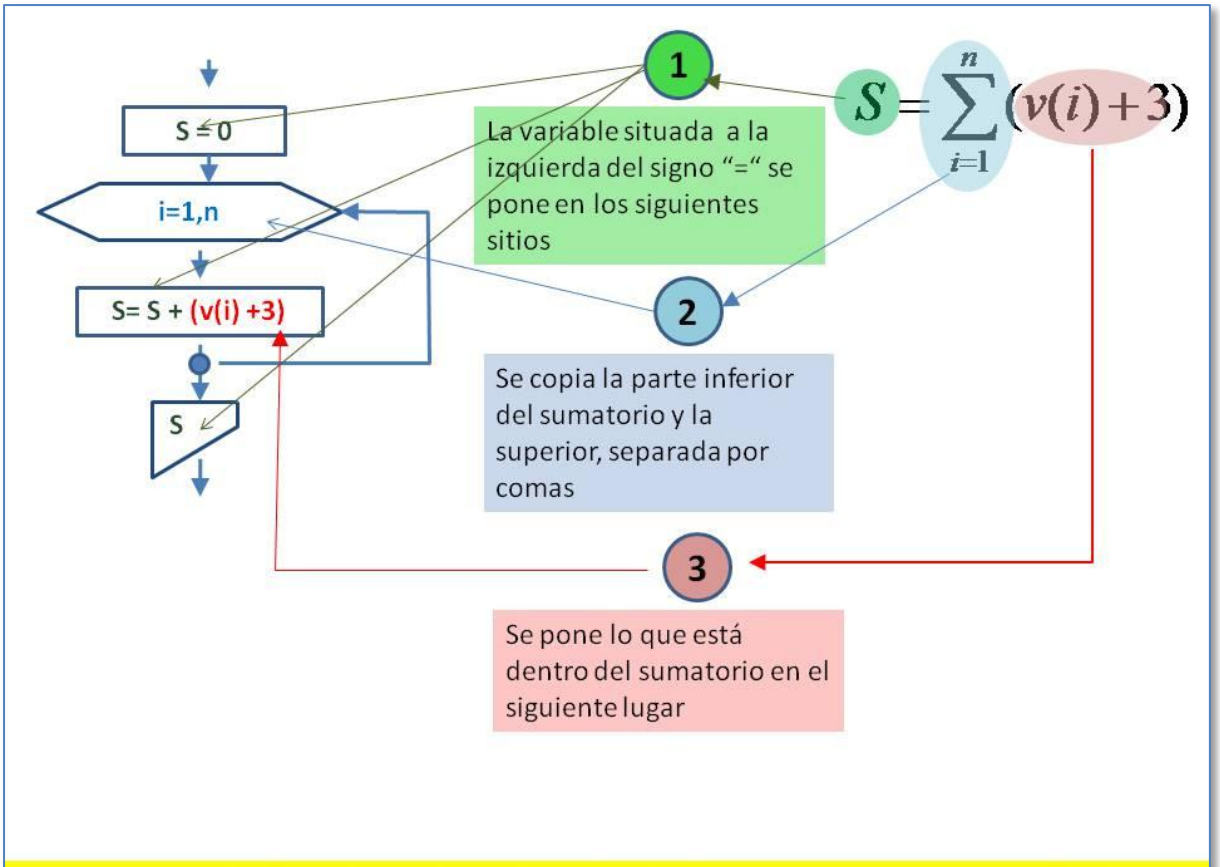

Figura 7. Método para hacer el algoritmo de cualquier sumatorio (no se incluye lectura de datos)

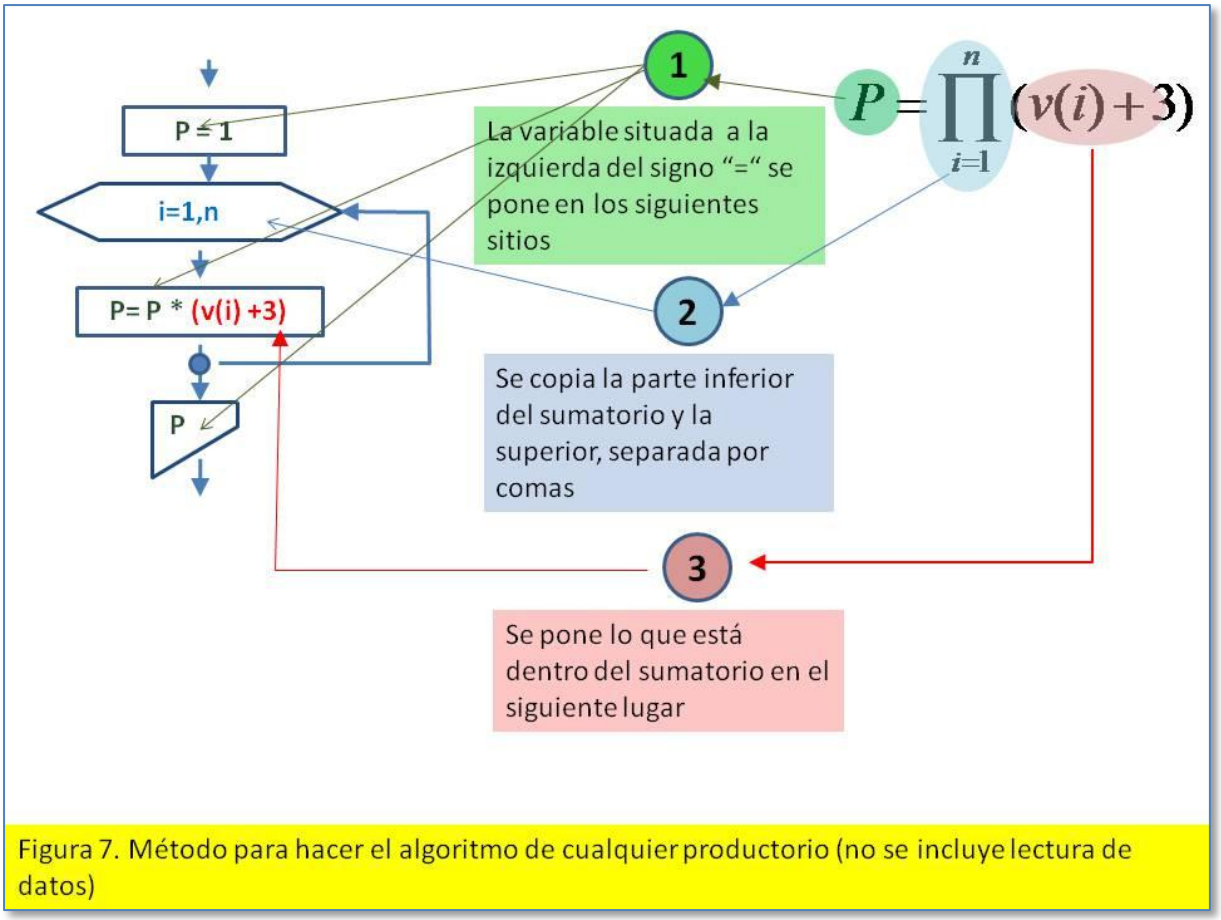

Figura 7. Método para hacer el algoritmo de cualquier productorio (no se incluye lectura de datos)

4.2.2. Bucles While.

4.2.2.1. Concepto del bucle While.

El bucle **While** se utiliza cuando realmente no sabemos el número de iteraciones que se van a realizar en el bucle, sin embargo sí sabemos que mientras se cumpla una determinada condición debemos realizar el bucle. Cuando esa condición ya no se cumpla entonces el bucle finalizará.

En este tipo de bucles no existe ninguna variable que indique el número de vueltas que ha realizado o falta por realizar. El único control que utiliza es una condición que está al comienzo del bucle, mientras esta condición sea cierta se realizan iteraciones y en el momento que sea falsa se finaliza el bucle.

Obsérvese que la condición está al comienzo del bucle y ésta hace las veces de una puerta de entrada al mismo. La puerta se abre si la condición es verdadera y la puerta se cierra si la condición es falsa.

Al estar la condición a la entrada del bucle, lo primero que se hace es comprobar la condición y después la iteración; esto significa que antes de comenzar a realizarse el bucle debe comprobar que la condición es verdadera, si ésta no lo es entonces el bucle no se realizará. De hecho cada vez que este bucle hace una iteración debe volver a pasar por la puerta y comprobar si la condición es verdadera o falsa.

Este tipo de bucles es muy útil en programación puesto que simula procesos reales; por ejemplo en un surtidor se podría llenar los depósitos de combustible mientras haya gasolina; una máquina expendedora de billetes puede vender billetes mientras haya billetes. En general este bucle tiene aplicación cuando nos encontremos ante un problema que se enuncie de la forma **mientras..ocurra algo...hacer tal cosa**. El "ocurra algo" es la condición del bucle y "hacer tal cosa" es el cuerpo del bucle; es decir, las sentencias que se ejecutan durante la iteración. Por ejemplo:

Como ya sabes, las máquinas de tabaco suministran un paquete determinado introduciendo unas monedas. Esta máquina funciona con un bucle de tipo While ya que se podría expresar por: "**mientras haya paquetes de tabaco venderlos**".

mientras(haya tabaco)

venderlos.

La condición "**haya tabaco**" se puede expresar por **(tabaco >0)** y el proceso "**venderlos**" por: **tomar dinero; dar cajetilla; devolver cambio** (si lo hay) **y hacer tabaco=tabaco-1** (quiere decir que queda una cajetilla menos). Así se podría expresar:

mientras (tabaco>0)
tomar dinero.
dar cajetilla.
devolver cambio (si lo hay).
tabaco=tabaco-1. (se ha vendido un paquete).
fin del bucle.

Cuando se utiliza este tipo de bucles se ha de tener en cuenta dos cosas:

1.- Al comienzo del programa los valores de la expresión que forma la condición están inicializados. En el ejemplo anterior es lógico que antes de poner a funcionar la máquina hayan metido cajetillas de tabaco. Si por ejemplo, se han metido inicialmente 100 cajetillas en el programa se expresa "**tabaco=100**" y esto se realiza antes de comenzar el bucle.

2.- Dentro del bucle se altera la condición de entrada; es decir, si se está ejecutando el bucle esto quiere decir que la condición es verdadera. Si esta condición no cambiara nunca entonces el bucle sería infinito (ya que la puerta estaría siempre abierta). Así pues, parece lógico suponer que dentro del bucle llegará un momento en el que la condición sea falsa y por tanto finalice el bucle. En el ejemplo anterior existe la sentencia "**tabaco=tabaco-1**" que causará que la condición (tabaco>0) sea falsa. Realmente esto ocurrirá cuando se realicen 100 iteraciones dentro del bucle o lo que es lo mismo cuando se hayan vendido las 100 cajetillas de tabaco.

Recuerda que: los bucles while antes de comenzar a realizar una iteración comprueban una condición, si ésta es verdadera entonces se realiza una iteración y si es falsa entonces finaliza la ejecución del bucle. Esto suele ser propenso a construir bucles que no se ejecutan nunca o bucles que se ejecutan indefinidamente (bucles infinitos).

Los bucles que no se ejecutan nunca se suelen presentar cuando antes del bucle la condición es falsa, por tanto no se entra en el bucle y éste no se realiza. Si ocurre esto revisa los valores iniciales que conforman la expresión que hay dentro de la condición.

Los bucles infinitos se suelen presentar cuando dentro del bucle no se cambia la condición de entrada. Si inicialmente esta condición es verdadera entonces se entra en el bucle; pero si dentro del mismo nunca se hace la condición falsa, entonces el bucle se ejecuta indefinidamente.

4.2.2.2. Representación algorítmica del bucle While.

La estructura general de la sentencia while es la siguiente:

(1)mientras (condición)
(2) bloque de sentencias;
(3)fin while.

donde **bloque de sentencias** es cualquier conjunto de sentencias C. La estructura tiene tres partes:

(1) **mientras condición** se representa por el mismo símbolo que una sentencia **if** y dentro de él la condición.
(2) cada sentencia se representa por su símbolo.
(3) el **fin while** se representa mediante un punto.

La **figura nº 8** expresa un organigrama de la sentencia While y su explicación.

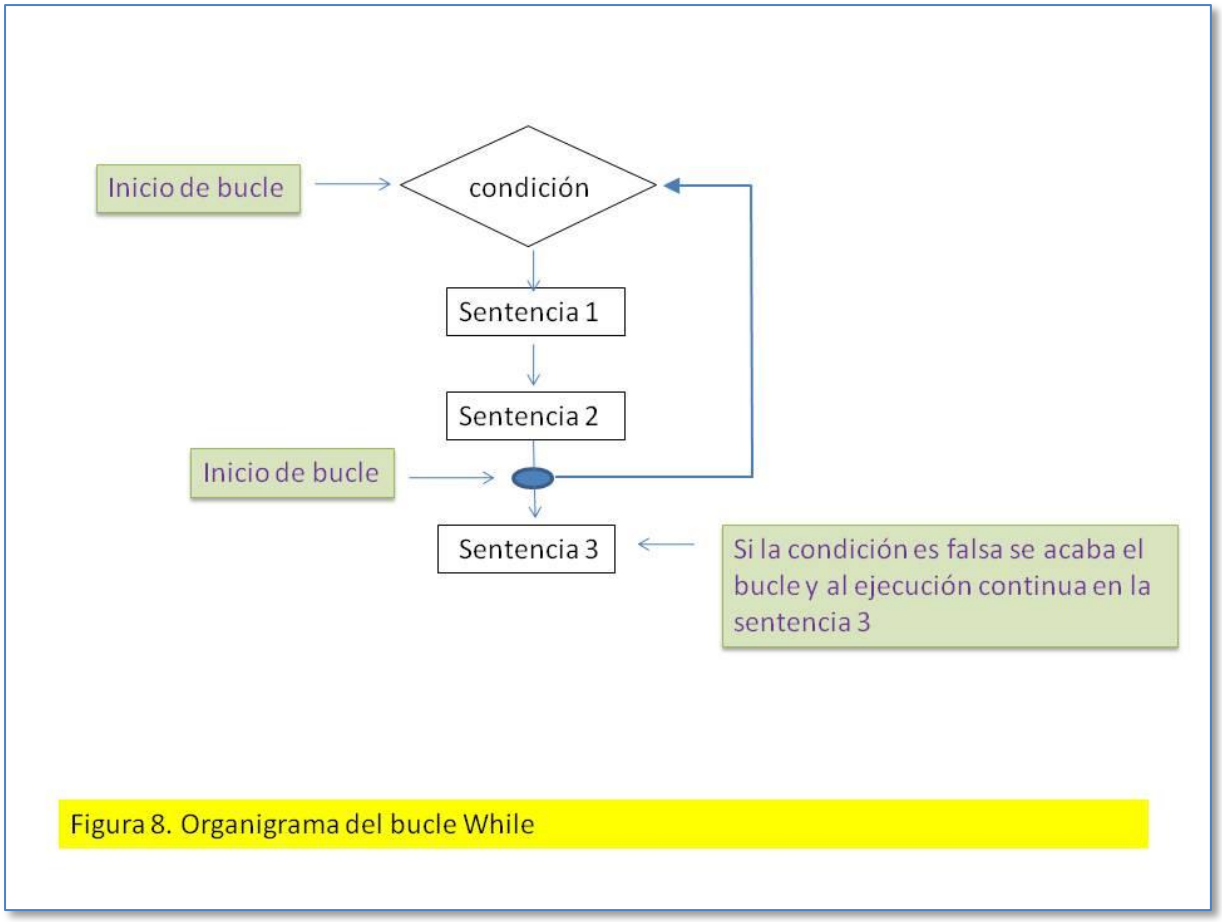

Figura 8. Organigrama del bucle While

Obsérvese que el inicio del bucle es idéntico al inicio de una sentencia if. Si la condición es verdadera entonces se ejecutan las sentencias del bloque, mientras que si la condición es falsa entonces la próxima sentencia a ejecutar es la que esté detrás del **fin while**.

Recuérdese que dentro del bloque de sentencias debe haber alguna que cambie la condición de entrada al mismo. Para comprender mejor el funcionamiento de este tipo de bucle realizaremos el siguiente ejemplo:

"Realizar un organigrama que represente el funcionamiento de una máquina de tabaco (por simplificar, suponer que sólo hay una marca de tabaco con un precio fijo). La máquina debe aceptar el dinero, dar el paquete y devolver el cambio".

Este problema se puede resolver con la estructura general: "mientras haya tabaco
venderlo", dónde venderlo es un conjunto de acciones. Esta estructura indica que el organigrama se puede realizar a través de un bucle While.

La **figura nº 9** muestra la solución.

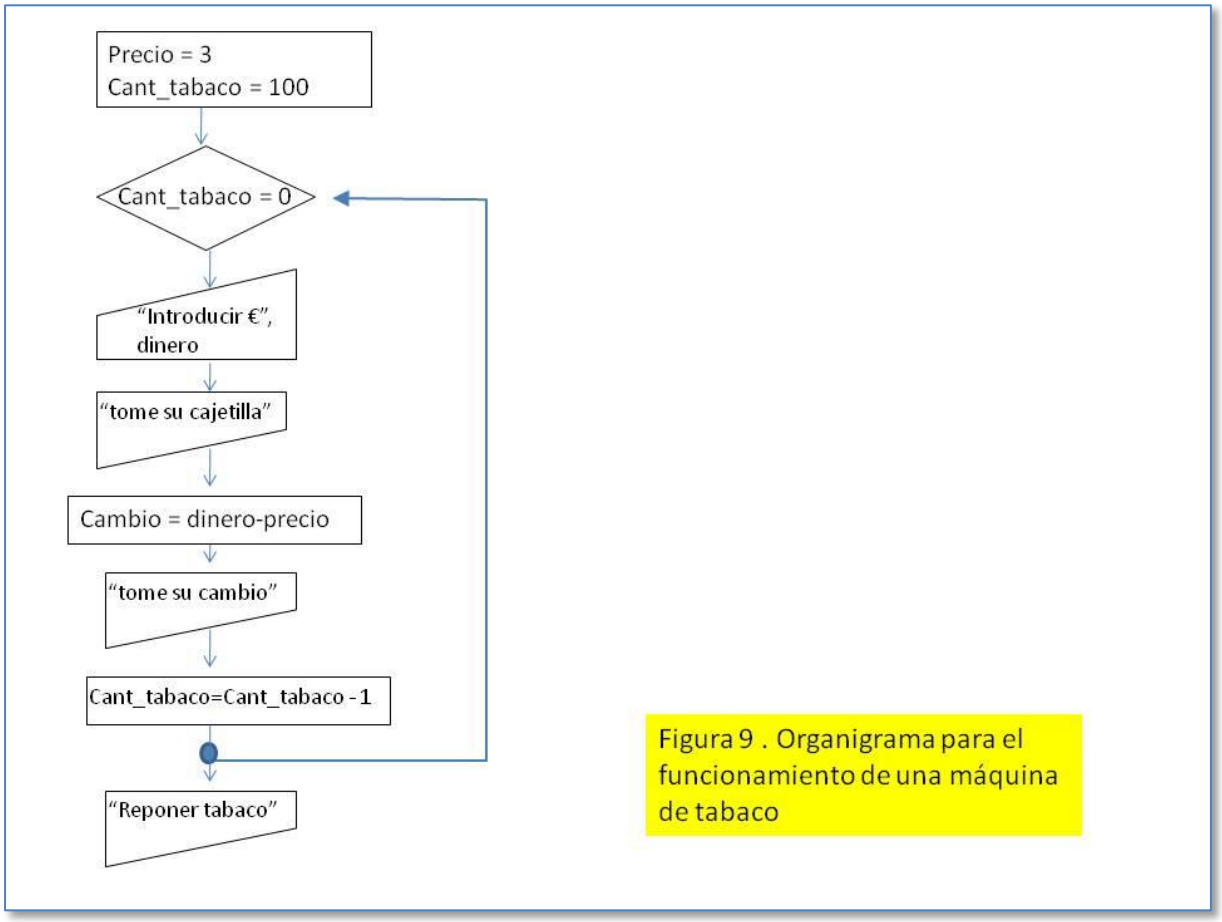

Figura 9 . Organigrama para el funcionamiento de una máquina de tabaco

4.2.3. Bucles Do-While.

4.2.3.1. Concepto del bucle Do-While.

Los bucles **for** y **while** tienen la característica común de que la condición se evaluaba antes de realizar el bucle, así cabía la posibilidad de que no se realizara ninguna iteración. La principal diferencia del bucle **do-while**, respecto a los anteriores, es que éstos evalúan la condición al final del bucle; así el bucle siempre realizará al menos una iteración.

En los bucles **do-while** no se sabe a priori el número de iteraciones que se van a realizar, depende de la condición que hay al final del bucle; si ésta es cierta se vuelve a realizar una iteración, mientras que si es falsa se acaba la ejecución del bucle.

La condición al final del bucle actúa como una compuerta de salida del mismo, si la condición es verdadera esta compuerta estará cerrada y por tanto debemos realizar una nueva iteración. Si la condición es falsa, entonces la compuerta estará abierta y se podrá abandonar la ejecución del bucle. En este tipo de bucle no hay ninguna variable que controle el número de iteraciones, el control lo realiza la condición, así pues es lógico suponer que dentro de las sentencias que componen el bucle haya al menos una que pueda cambiar el valor de la condición o compuerta de salida.

Al igual que los bucles **while** este tipo de bucles es muy utilizado puesto que muchas situaciones reales tienen una interpretación directa en el bucle **do-while**. Concretamente todas aquellas que se puedan expresar mediante: **"haz .....acciones.....mientras ocurra una condición"** la estructura del bucle sería:

```
hacer
bloque de sentencias
mientras (condición)
```

Obsérvese que **mientras (condición)** está al final del bucle y esto hace que el bloque de sentencias se realice al menos una vez (se realiza una iteración) independientemente del valor que tenga la condición. Esta característica es la que determina la utilización del bucle; es decir, **si debemos realizar un bucle en el cual es necesario al menos hacer una iteración, entonces utilizaremos el bucle do-while.**

Son muchas las situaciones en las cuales se requiere que el bucle se

ejecute al menos una vez, por ejemplo **todas aquellas que necesiten que una entrada de información cumpla una serie de condiciones y no se quiera continuar con el programa hasta que se cumplan**. Por ejemplo, en el organigrama de la máquina de tabaco se introducía una cantidad de dinero para adquirir un paquete, sería conveniente no dejar continuar la ejecución del programa hasta que la cantidad de dinero introducida no fuese igual o superior al precio del paquete de tabaco. Esto se realizaría así:

```
haz
leer precio (cantidad introducida)
mientras (precio<200)
```

Este bucle **do-while** no dejaría continuar con el programa hasta que el usuario introdujera al menos 200 pesetas. Si intentamos hacer esta situación con un bucle **while** tendríamos:

```
precio=0                    ó   leer precio
mientras (precio<200)   mientras(precio<200)
leer precioleer precio
fin del bucle.fin del bucle.
```

Observa que en el primer caso con la sentencia **precio=0** estamos forzando a que el bucle **while** se realice al menos una vez, esta situación se resuelve mejor con un bucle **do-while**; en el segundo caso necesitamos dos sentencias de lectura para realizar lo mismo que el bucle **do-while** con una sola sentencia.

Estas situaciones resaltan la necesidad de los bucles **while** de inicializar las variables que intervienen en la condición, mientras que los bucles **do-while** no tienen necesidad de inicializar ninguna variable.

A continuación veamos un ejemplo de utilización de bucles **do-while** :

"Realizar un organigrama a través del cual se pueda jugar al número escondido. En este juego alguien escribe un número y otra persona trata de adivinarlo, esta persona dirá un número y si es mayor que el que está escrito se deberá decir "alto" mientras que si es menor se deberá decir "bajo". El juego finaliza cuando se acierta el número escrito y gana aquel que lo haya acertado en menos intentos".

Este juego se puede expresar de la forma: "di un número mientras no

aciertes el número secreto" ó "juega mientras no aciertes el número secreto". Esto se puede expresar mediante un bucle **do-while** de la siguiente forma:

```
x=0
haz
lee número (el jugador dice un número)
si es mayor que el secreto escribe ("alto")
si es menor que el secreto escribe ("bajo")
x=x+1
mientras (número distinto al secreto)
escribe ("has acertado en ", x , "intentos")
```

Observa que el bucle se realiza hasta que el jugador acierta el número secreto. La variable x mide el número de intentos que ha realizado el jugador hasta acertar.

4.2.3.2. Representación algorítmica del bucle Do-While.

La estructura del bucle **do-while** tiene tres partes diferenciadas: inicio, bloque de sentencias y final.

```
do(comienzo del bucle).
bloque de sentencias;(bloque de sentencias).
mientras (condición)(fin del bucle).
```

En este caso la condición final del bucle también actúa como "compuerta" de salida o indicador de iteraciones. El organigrama del bucle **do-while** se representa en la **figura nº 10**

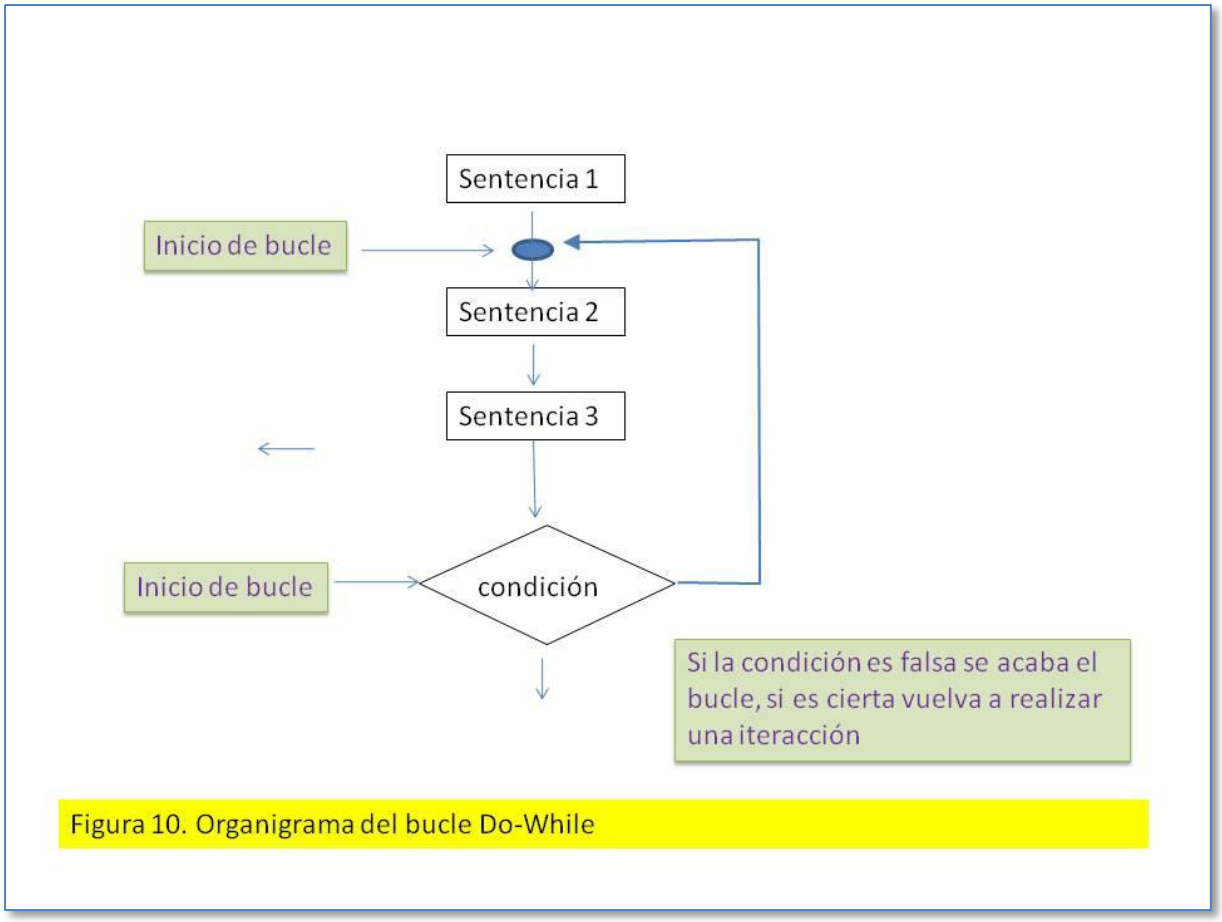

Figura 10. Organigrama del bucle Do-While

Obsérvese que el final del bucle (mientras (condición)) se representa por el mismo símbolo que una sentencia **if**. Dentro de las sentencias que componen el cuerpo del bucle debe haber alguna que posibilite cambiar el valor de la condición, ya que si el valor de ésta inicialmente es verdadero, entonces realizaríamos un bucle infinito.

El algoritmo a través del cual representábamos el juego del número secreto se representa mediante un organigrama como se muestra en la **figura nº 11**

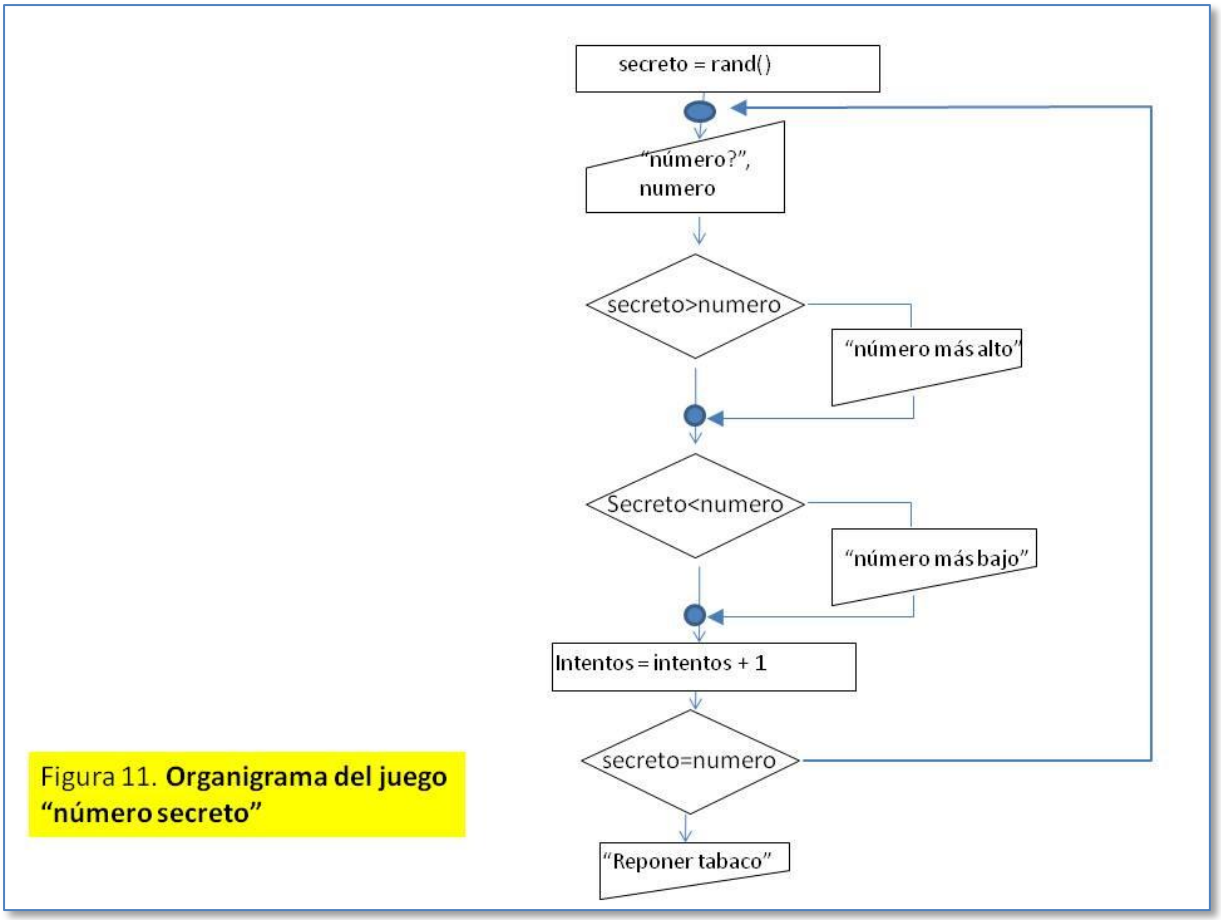

Figura 11. **Organigrama del juego "número secreto"**