TEMA 4
LA ITERACIÓN

1 MOTIVACIÓN

Como ya se ha dicho en los temas anteriores, programar consiste en dar solución a un problema a través de una secuencia de instrucciones básicas que el ordenador sea capaz de ejecutar. Las iteraciones forman parte del grupo de instrucciones básicas que un ordenador puede entender. A través de las iteraciones podremos reducir un cierto problema complejo convirtiéndolo en la repetición de acciones más simples hasta que se cumpla una determinada condición.

1.1 Porqué y para qué **utilizar la iteración?**

Una razón puede ser la que ya hemos aludido. **La repetición de ciertas acciones simples nos puede llevar a la solución de un problema más complejo.** Vamos reflexionar sobre lo dicho a través de un ejemplo muy sencillo. Como es bien sabido una división es una operación matemática que reviste mayor complejidad que una suma (de hecho en la escuela nos enseñan antes a sumar que a dividir…). El ejemplo consiste en solucionar una división a través del la iteración, o dicho con otras palabras, la repetición de operaciones de suma. Es decir, vamos a solucionar una operación más compleja a través de la repetición de una operación más sencilla.

1.2 Ejemplos

1.2.1 Ejemplo 1

Supongamos que tenemos que solucionar la siguiente división entera 347/111 (sólo lo haremos aplicando sumas y sin necesidad de saber las tablas de multiplicar).

0+111=111 (**una suma**) +111 = 222 (**dos sumas**) + 111 = 333 (**tres sumas**) + 111 = 444. El 444 supera 347 así que está última suma nos hace ver que ya no tenemos que seguir sumando, la solución es 3 es decir tantas sumas como hemos tenido que realizar **mientras el número que íbamos obteniendo era menor que 347**.

El objetivo de este ejemplo es ver como a través de la **repetición** de **acciones más simples MIENTRAS** una determinada condición sea cierta, nos pueden llevar a las resolución de algo más complejo.

1.2.2 Ejemplo 2

Vamos a plantear un ejercicio (en este caso el cálculo de una nota media) e inicialmente vamos a tratar de resolverlo sin utilizar la iteración. Analizaremos los problemas que nos surgen y en el punto 4.2.3.3 daremos la solución utilizando la iteración. Veremos que las solución iterativa es la solución adecuada, sencilla, corta y generalizable.

Supongamos que en una escuela nos dicen que quieren un programa que les calcule dada una clase la nota media obtenida por los alumnos de esa clase. A priori

no sabemos cuantos alumnos va a tener la clase. En algunos casos tendrá 30 en otros 50 en otros 100 etc. Sabemos que para calcular la media tenemos que sumar todas las notas de los alumnos y dividir la *suma acumulada* entre el *número de alumnos*. Así pues iremos pidiendo al profesor que vaya introduciendo las notas de sus alumnos una a una, de forma que cada nota la iremos sumando a las notas acumuladas hasta el momento, y a la vez por cada suma iremos incrementando un contador de alumnos, es decir iremos contabilizando los alumnos a medida que se van introduciendo sus notas. Así podremos calcular la suma_acumulada_de_notas y el número_de_alumnos.

Versión NO ITERATIVA (es decir, utilizando sólo las instrucciones que conocemos hasta el momento)

Cuántas variables precisaremos?

1er problema: La declaración de las variables será nuestro primer problema.

Como el número de alumnos nos será desconocido a la hora de declarar las variables (sólo en ejecución podríamos saberlo…) no sabremos CUANTAS tenemos que declarar, cuantas notas meteremos? Alguien puede pensar, *"bueno pues hacemos una estimación y estimamos que como mucho tendremos 100"*. Bueno, esto podría considerarse como una solución. Muy tediosa porque ¡debeberemos escribir 100 VARIABLES!!!

```
float nota_1, nota_2, nota_3, nota_4, nota_5,     así hasta  nota_100
 /*En el ordenador  tendremos que escribirlas todas*/
```

Ahora nos quedaría declarar 3 variables más 1 para ir acumulando las notas; *suma acumulada*, otra para ir contando cuantos alumnos tendremos: *número de alumnos*(una vez que el profesor haya introducido todas las notas lo sabremos porque por cada nota introducida sumaremos 1 a esta variable). Y por último la variable (*media*) en la que guardaremos el valor de la división *suma_acumulada/número_de_ alumnos* es decir la media.

```
float suma_acumulada, numero_de_alumnos, media; /*cuidado con las tildes */
```

Parece que hemos superado el primer escollo (¿cuántas variables?), aunque la solución propuesta sea muy pesada y poco práctica por ser tan larga y por utilizar tanta memoria. Como veremos en el punto 4.2.3.3 con el uso de la iteración el número de variables quedará reducida de 103 a 4).

Después de declaradas las variables, nuestro programa podría empezar por inicializar las variables que tienen que ser inicializadas.

```
numero_de_alumnos = 0; /*al principio y hasta introducir algún dato el número de alumnos será 0*/
suma_acumulada = 0; /*Esta variable también y hasta no introducir ningún dato tendrá valor 0*/

printf ("Introduce una nota por favor: ");
scanf("%f",&nota_1);
numero_de_alumnos = numero_de_alumnos+1; /*Ahora y como hemos pedido una nota
                                          el número de alumnos pasará de 0 a 1, es decir 0+1=1*/
suma_acumulada  = suma_acumulada + nota_1; /*La suma acumulada también la tendremos que
                                          actualizar ya no será 0 tampoco, 0+ nota_1=nota_1*/

printf ("Introduce una nota por favor: ");
scanf("%f",&nota_2);
numero_de_alumnos = numero_de_alumnos+; /*otra vez actualizamos el número de alumnos sumando
```

```
                                        uno más */
suma_acumulada = suma_acumulada + nota_1; /*La suma acumulada también la tendremos que
                                        actualizar acumulando la nueva nota*/

printf ("Introduce una nota por favor: ");
scanf("%f",&nota_3);
numero_de_alumnos = numero_de_alumnos+; /*otra vez actualizamos el número de alumnos sumando
                                        uno más */
suma_acumulada = suma_acumulada + nota_3; /*La suma acumulada también la tendremos que
                                        actualizar acumulando la nueva nota*/
printf ("Introduce una nota por favor: ");
scanf("%f",&nota_4);
numero_de_alumnos = numero_de_alumnos+; /*otra vez actualizamos el número de alumnos sumando
                                        uno más */
suma_acumulada = suma_acumulada + nota_4; /*La suma acumulada también la tendremos que
                                        actualizar acumulando la nueva nota*/
```

1. Problema

El mismo problema vuelve a resurgir. ¿Cuántas notas tenemos que pedir?, como no sabemos cuantos alumnos tendremos ¿cómo decidiremos como parar?¿Para que pedir 100 si quizás no haya tantos alumnos? Una solución podría consistir en establecer un código con el profesor (el usuario) de forma que si el introduce por ejemplo un −1 sabremos que ya no quiere introducir más notas, es decir, que su lista se ha acabado. Le escribiríamos un mensaje al comienzo del programa para que fuese consciente de que debe marcar con un −1 el final de su lista. Por ejemplo, printf ("Cuando introduzcas un −1 como nota significará que ya no hay más notas para introducir");

Ahora tendremos que modificar nuestro programa para que no le pida al profesor que introduzca más notas una vez que éste haya pulsado −1.

```
printf ("Introduce una nota: ");
scanf("%f",&nota_1);
if (nota_1 != -1){
        numero_de_alumnos = numero_de_alumnos+;
        suma_acumulada = suma_acumulada + nota_1;

        printf ("Introduce una nota: ");
        scanf("%f",&nota_2);
        if (nota_2 != -1){
                numero_de_alumnos = numero_de_alumnos+;
                suma_acumulada = suma_acumulada + nota_2;

                printf ("Introduce una nota:");
                scanf("%f",&nota_3);
                if (nota_3 != -1){
                        numero_de_alumnos = numero_de_alumnos+;
                        suma_acumulada = suma_acumulada + nota_3;
                        printf ("Introduce una nota: ");
                        scanf("%f",&nota_4);
                        if (nota_4 != -1){
                                numero_de_alumnos = numero_de_alumnos+;
                                suma_acumulada = suma_acumulada + nota_3;
                                printf ("Introduce una nota: ");
                                scanf("%f",&nota_5);

                        /*…… y así 95veces más .
                        al ordenador le tendremos que escribir las 95 veces todo no entiende los
```

```
                                    puntos suspensivos */

                                    }
                            }
                    }
            }
}

if (numero_de_alumnos !=0){
            media= suma_de_notas/numero_de_alumnos;
  }
  else{
       media= 0;
       }
```

Otra vez parece que hemos conseguido solucionar el problema de cómo y cuando acabar sin que el profesor tenga necesariamente que introducir 100 notas. A medida que va introduciendo notas vamos avanzando siempre y cuando lo tecleado no sea un –1. Otra vez, con la iteración veremos en el punto 4.2.3.3 que se puede solucionar de una forma mucho mas corta y sencilla, más aun teniendo en cuenta que todo el rato estamos repitiendo casi las mismas instrucciones.

```
numero_de_alumnos = numero_de_alumnos+ 1;
suma_acumulada = suma_acumulada + nota_????;
printf ("Introduce una nota: ");
scanf("%f",&nota_????);
```

Después de explicar la sintaxis y el significado de la iteración, volveremos a hacer este mismo ejercicio pero esta vez de forma iterativa.

Con las instrucciones que hemos visto hasta ahora (2º tema: operaciones aritméticas, la asignación, instrucciones de entrada y salida), las instrucciones condicionales (3er tema), junto con las instrucciones iterativas **YA SEREMOS CAPACES DE SOLUCIONAR CUALQUIER PROBLEMA QUE SE NOS PLANTEE. Programar consistirá en encontrar la combinación correcta de estas instrucciones para dar una solución.**

2 LA ITERACIÓN EN C

La iteración siempre irá asociada a una condición de terminación. Es decir, **se iterarán (se repetirán) unas determinadas instrucciones MIENTRAS UNA CIERTA CONDICIÓN SEA VERDAD**. Las condiciones asociadas a una iteración son de la misma forma que aquellas que hemos visto para las instrucciones condicionales (if, if-else, ...) . Así utilizaremos para generar condiciones de iteración los operadores relacionales (==, <, >, <=, >=) y los operadores lógicos (!, &&, ||). La iteración nos puede aparecer bajo 3 distintas formas en C:

1. **While** (condición){
   Conjunto de instrucciones que han de repetirse
   }.

2. **Do**{
   Conjunto de instrucciones que han de repetirse
   }**while** (condición);

3. **For** (inicialización ; condición ; actualización){
   Conjunto de instrucciones que han de repetirse

}

Las tres son muy parecidas pero la más genérica es el **while**, que se puede utilizar en cualquier caso. Las otras dos tienen usos más restringidos.

3 EL WHILE

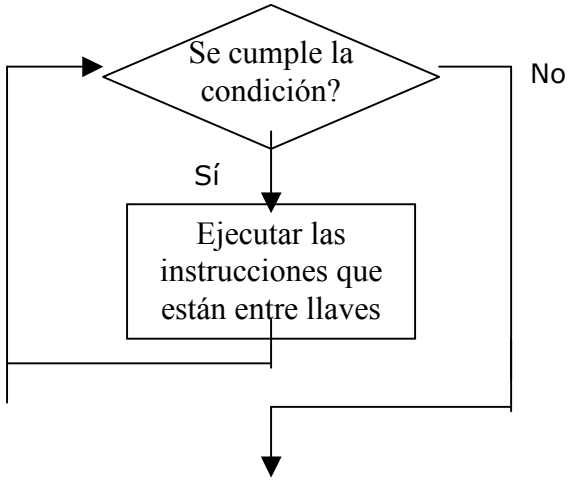
- Su forma:

```
while (condición)
 {
   Conjunto de instrucciones que han de
repetirse
 }
```

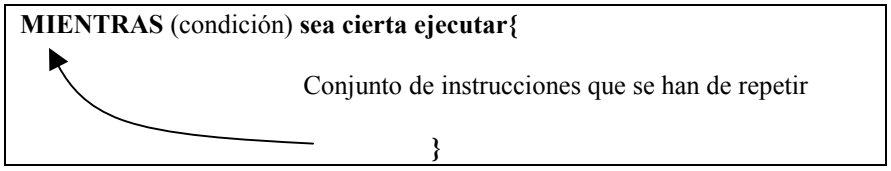
- Su significado:

Mientras la condición sea cierta ejecutar las instrucciones que aparecen entre llaves.
Estas instrucciones se ejecutarán una y otra vez hasta que llegue un momento en el que la condición sea falsa. Por eso deberemos verificar que entre una ejecución y otra de esas instrucciones algo cambie de forma que la condición pueda llegar a ser falsa alguna vez.

- Su significado gráficamente:



- El while en castellano:



Insistimos en que es preciso que cambie algo de una vuelta a la otra, sino entraríamos en lo que se llama un **BUCLE INTERMINABLE**. Es decir, una iteración que no acaba nunca.

Por ejemplo,

```
a=0;
while(a<10){
      printf (“%d”,a);
}
```

esto sería un **bucle interminable**. El programa asigna valor 0 a una variable a y luego pasa a mirar si la condición del while es cierta. Como lo es, entra en las llaves y ejecuta el printf. Una vez ejecutado vuelve a comprobar si la condición sigue siendo cierta. Como lo es, vuelve a entrar, imprime el valor de a y vuelve a comprobar la condición…. Y así eternamente porque la condición **NUNCA dejará de ser cierta** mientras dentro de las llaves de while no cambie el valor de a. Si a **SIEMPRE vale 0 a SIEMPRE será <10 y por lo tanto la condición SIEMPRE será VERDAD**.

Sin embargo comparemos con este código

```
a=0;
while(a<10){
      printf (“%d”,a);
               a= a+1;
}
```

al comienzo a tiene el valor 0 y como se cumple la condición de a<10 entra dentro de las llaves y ejecuta lo que haya dentro. Es decir, imprime el valor de a y LUEGO LE SUMA 1, haciendo que el valor de a se incremente (¡¡HA CAMBIADO!!), así cuando volvemos a evaluar la condición a vale 1 pero aun así a <10 con lo cual volvemos a entrar de nuevo. imprime el valor de a y LUEGO LE SUMA 1, haciendo que el valor de a se incremente de nuevo, ahora valdrá 2 (¡¡HA VUELTO A CAMBIAR!!). Ahora si que vemos que el valor de a se va modificando en cada vuelta de forma que en la 9ª vuelta tomara el valor 10 con lo cual al volver a evaluar la condición ÉSTA DEJA DE SER CIERTA y ya no entraremos más en las llaves. NO HAY BUCLE INTERMINABLE ESTA VEZ.

3.1 Un ejemplo simple: Escribir 5 veces un mensaje

Escribir un programa tal que escriba 5 veces en pantalla el mensaje Kaixo!! Egunon!!.

Una forma de hacerlo consistiría en utilizar 5 printf-s. Pero esta no sería la mejor solución ya que si nos dicen que hemos de cambiar el programa de forma que escriba el mensaje 1000 veces tendríamos que utilizar ¡¡¡1000 printfs!!!!. ¡¡¡Una locura!!!. Utilizaremos un único printf en conjunto con una instrucción iterativa.

```
#include <stdio.h>

void main()
 {   int cuantas_veces;


   cuantas_veces = 0; /* Esta variable nos permitirá computar cuantas veces
                      vamos imprimiendo el mensaje por lo tanto inicialmente -
…….valdrá 0*/

   while (cuantas_veces < 5){
      printf("\nKaixo! Egunon!");
      cuantas_veces = cuantas_veces + 1;
    }

 }
```

¿Cuándo acabará el programa?

Cuando la condición (*cuantas_veces* < 5)deje de ser cierta, en este caso después de 5 vueltas, ya que en cada vuelta vamos incrementando en 1 el valor de la variable. Por lo tanto harán falta 5 vueltas para que *cuantas_veces* tome el valor 5.

A continuación mostramos la simulación de la ejecución del programa:

- int `cuantas_veces`; **/*así reservaremos espacio de memoria para la variable*/**

  `cuantas_veces` | ?

  Pantalla

- cuantas_veces = 0;

  cuantas_veces | 0

  Pantalla

- cuantas_veces < 5 ? SÍ

  cuantas_veces | 0

  Pantalla

- printf("Kaixo! Egunon!");

  cuantas_veces | 0

  Pantalla | Kaixo! Egunon!

- cuantas_veces = cuantas_veces + 1;

  cuantas_veces | 1

  Pantalla | Kaixo! Egunon!

- cuantas_veces < 5 ? SÍ

cuantas_veces | 1 |

Pantalla

```
Kaixo! Egunon!
```

- printf("Kaixo! Egunon!");

cuantas_veces | 1 |

Pantalla

```
Kaixo! Egunon!
Kaixo! Egunon!
```

- cuantas_veces = cuantas_veces + 1;

cuantas_veces | 2 |

Pantalla

```
Kaixo! Egunon!
Kaixo! Egunon!
```

- cuantas_veces < 5 ? SÍ

cuantas_veces | 2 |

Pantalla

```
Kaixo! Egunon!
Kaixo! Egunon!
```

- printf("Kaixo! Egunon!");

cuantas_veces | 1 |

Pantalla

```
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
```

- cuantas_veces = cuantas_veces + 1;

cuantas_veces | 3 |

Pantalla

```
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
```

- cuantas_veces < 5 ? SÍ

cuantas_veces | 3 |

Pantalla

```
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
```

- printf("Kaixo! Egunon!");

cuantas_veces | 3 |

Pantalla

```
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
```

- cuantas_veces = cuantas_veces + 1;

cuantas_veces | 4 |

Pantalla

```
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
```

- cuantas_veces < 5 ? SÍ

cuantas_veces | 4 |

Pantalla

```
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
```

- printf("Kaixo! Egunon!");

cuantas_veces | 4 |

Pantalla

```
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
```

- cuantas_veces = cuantas_veces + 1;

cuantas_veces | 5 |

Pantalla

```
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
```

→

- cuantas_veces < 5 ? **YA NO**

cuantas_veces | 5 |

Pantalla

```
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
```

cuantas_veces | 5 |

Pantalla

```
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
Kaixo! Egunon!
.
```

3.1.1 Programa que presenta en pantalla los primeros n números naturales

El programa deberá pedir al usuario que introduzca un número n tal que (n≥ 1 y n<=10). El programa deberá imprimir los número desde 1 hasta n. Si el usuario introdujese un valor para n que no cumplise la condición entonces se le escribirá al usuario un mensaje diciéndole que el valor de n no es correcto y el programa acabará.

```
#include <stdio.h>

void main()
 {
   int n, num;

 printf("\nIntroduce un número entre 1 y 10 (1<= número <=10): ");
 scanf("%d", &n);

   if ((n < 1)||(n > 10))
      printf("\nEl número no es adecuado.");
    }
   else
    {

      num = 1; /* p.q. en la primera vuelta del while queremos imprimir el 1 */
      while (num < n)
       {
         printf("\n%d", num);
         num = num + 1; /* incrementamos el valor de num para que en la siguiente
                           vuelta del while valga uno más */
```

```
       }/*final del while, es decir final de las actciones a repetir*/
    }/*el final del else*/
}
```

3.1.2 Suma de los primeros n números naturales

El programa deberá pedir al usuario que introduzca un número n tal que (n≥ 1 y n<=10). El programa deberá imprimir la suma de dichos números.

Si el usuario introdujese un valor para n que no cumplise la condición entonces se le escribirá al usuario un mensaje diciéndole que el valor de n no es correcto y el programa acabará.

```
#include <stdio.h>

void main()
 {
   int n, num, suma_acumulada;

 printf("\nIntroduce un número entre 1 y 10 (1<= número <=10): ");
 scanf("%d", &n);
   if (n < 1)
    {
      printf("\nEl valor para n no es apropiado.");
    }
   else
    {
      suma_acumulada = 0;
      num = 1;
      while (num <= n)
       {
         suma_acumulada = suma_acumulada + num;
         num = num + 1;        }
      printf("\nLehenengo %d zenbaki naturalen sumando %d da.", n, sumando);
    }
   printf("\nSakatu tekla bat amaitzeko.");
 }
```

3.1.3 Nuestro ejemplo inicial: el cálculo de la media

Ahora que ya sabemos calcular la suma de todos los números hasta un determinado número n, podemos replantearnos el ejemplo de la media de forma iterativa. Recordemos que se nos pedía calcular la nota media de los alumnos de una clase.

En el ejemplo 2 del punto 4.1 intentamos resolver este problema sin utilizar la iteración. Como vimos en ese punto nos surgían varios problemas, cuantas variables utilizar, cuando y como acabar el programa … etc y aunque finalmente llegamos a dar una solución, el código resulto largo y tedioso, y ademas no funcionaría por ejemplo para clases de más de 100 alumnos.

Ahora que conocemos la sintaxis y el significado de la instrucción **while** vamos a replantear el ejercico utilizando la iteración.

La solución iterativa sería la siguiente:

```
#include <stdio.h>

void main()
 {
 float nota,suma_acumulada, numero_de_alumnos, media;
```

```
   suma_acumulada= 0; /* al comienzo y hasta no introducir ninguna nota
……..no habrá nada acumulado por lo tanto el valor será 0*/
   numero_de_alumnos= 0; /* Lo mismo con respecto al número de alumnos*/
    printf ("Introduce una nota por favor: ");
    scanf("%f",&nota);
     while (nota != -1){
        suma_acumulada= suma_acumulada+ nota;
        numero_de_alumnos = numero_de_alumnos+1;
        printf ("Introduce una nota por favor: ");
        scanf("%f",&nota);
     }
   if (numero_de_alumnos !=0){
              media= suma_acumulada/numero_de_alumnos;
   }
   else{
        media = 0;
        }
 }
```

Como se puede apreciar, el código es mucho más corto, necesitamos sólo 4 variables versus las 103 que necesitábamos en la versión no iterativa, y además el código resulta más genérico porque no queda limitado a 100 notas o un número concreto de notas sino que puede introducirse un número indeterminado de notas.

3.1.4 Presentar en pantalla los primeros n números pares

Este ejercicio consiste en pedir al usuario que introduzca un número n (1<=n<=10), una vez que tengamos un valor correcto para n nuestro programa deberá escribir los primeros n números pares.

Si el usuario introdujese un valor para n que no cumplise la condición entonces se le escribirá al usuario un mensaje diciéndole que el valor de n no es correcto y el programa acabará.

Para poder encontrar los n número pares una solución prodría consistir en ir generando números partiedo del 1 y por cada número comprobar si es par o no utilizando a la vez un contador para ir contando los números que sean pares[1].

El programa quedaría de la siguiente forma.

```
#include <stdio.h>

void main()
 {
   int n, num, cuantos;


   printf("\nIntroduce un número positivo(1<= numero <=10): ");
   scanf("%d", &n);

   if ((n < 1)||(n > 10))
    {
      printf("\nEl número que has introducido no es adecuado.");
    }
   else
    {
      cuantos = 0; /* inicialmente y como todavia no hemos encontrado ningún
```

[1] Existe más de una forma de solucionar este ejercicio. Por ejemplo partiendo como valor inicial de num=2 y en cada vuelta actualizarlo sumandole 2-> num=num+2. La solución que planteamos ha sido elegida por ser metodologicamente más adecuada para explicar el concepto de contador.

```
                    número par inicializaremos está variable a 0*/
      num = 1; /* inicializamos esta variable a 1 porque es el primer número
                  que queremos analizar */
      while (cuantos < n)
       {
         if (num % 2 == 0)
          {
            printf("\n%d", num);
            cuantos = cuantos + 1;/*esta variable es lo que llamamos
……….comúnmente contador porque va haciendo
……….recuento de cuantas veces sucede una
……….determinada cosa, en este caso encontrar
……….un número par*/
          }
         num = num + 1; /* en cada vuelta vamos incrementando num */
       }
    }

 }
```

3.1.5 Cuántas veces se puede dividir un número entre dos, sin generar un resto

En este ejercicio, hay que escribir el programa que decide cuántas veces se puede dividir un número entero positivo (≥ 1) entre el número 2 (sin generar un resto).

Hay que repetir el proceso de entrada de datos hasta conseguir un dato adecuado:

```
#include <stdio.h>

void main()
 {
   int n, naux, cont;

   printf("\nTeclee un valor entero positivo (>= 1): ");
   scanf("%d", &n);

   while (n < 1)
    {
      printf("\nEl valor tecleado no es adecuado.");
      printf("\nTeclee un valor entero positivo (>= 1): ");
      scanf("%d", &n);
    }

   naux = n; /* naux será una copia de n porque así cuando hagamos operaciones
……..sobre n no perderemos su valor inicial que quedará
……..intacto en naux. */

   cont = 0; /* Como está variable va a ser un contador que vaya contando
                cuantas veces el número n es divisible entre 2 al comienzo
               tendrá valor 0 */

   while (n % 2 == 0)
    {
      cont = cont + 1;
      n = n / 2;
    }

   printf("\nel número %d se puede dividir %d veces entre 2.", naux, kont);

 }
```

3.1.6 Presentar en pantalla y contar todos los divisores de un número dado n

En este ejercicio deberemos pedir al usuario un valor entero y positivo (n>0). Deberemos encontrar, presentar en pantalla y contar todos los divisores que pueda tener ese valor .

El proceso de petición del valor deberá acabar hasta que el usuario introduzca un valor adecuado.

```
#include <stdio.h>
void main()
 {
   int n, num, cont;

  printf("\nTeclee un valor entero positivo (>= 1): ");
  scanf("%d", &n);

   while (n < 1)
    {
      printf("\nEl valor tecleado no es valido.");
      printf("\nTeclee un valor entero positivo (>= 1): ");
      scanf("%d", &n);
    }

   num = 1; /* Utilizaremos esta variable para generar de uno en uno los
.....valores desde el 1 hasta n, así que tendremos que inicializarlo a
              1 porque este es el primer valor que hemos de generar*/

   cont = 0; /* Como está variable será un contador que vaya contando cuantas
                veces el valor que vaya tomando num será divisor de n, al
                comienzo su valor 0 hasta no encontrar el primer divisor*/

   printf("\nLos divisores de %d son: ", n);
   while (num <= n)
    {
      if (n % num == 0)
       {
         printf("\n%d", num);
         cont = cont + 1;
       }
      num = num+ 1;
    }
   printf("\n%d(e)k guztira %d zatitzaile ditu.", n, kont);
 }
```

3.1.7 Calculo de un sumatorio

En este ejercicio deberemos pedir al usuario un valor entero y positivo (n>0). Deberemos encontrar el valor del siguiente sumatorio

$$\sum_{i=1}^{n}((8*i)+2)$$

El proceso de petición del valor deberá acabar hasta que el usuario introduzca un valor :

```
#include <stdio.h>


void main()
 {
   int n, i, sumatorio;
```

```
  printf("\nTeclee un valor entero positivo (>= 1): ");
  scanf("%d", &n);

   while (n < 1)
    {
      printf("\nEl valor tecleado no es valido.");
      printf("\nTeclee un valor entero positivo (>= 1): ");
      scanf("%d", &n);
    }

   i = 1; /* el primer valor que tiene que tomar i es 1 por eso
             inicializaremos i con 1*/
   sumatorio = 0; /* En cada vuelta del while deberemos obtener un nuevo
                     sumatorio que deberemos ir acumulando en la variable
sumatorio.
                     Inicialmente lo acumulado será equivalente a 0 así pues
                     inicializaremos sumatorio a 0 */

   while (i <= n)
    {
      sumatorio = sumatorio+ ((8 * i) + 2);
      i = i + 1;
    }

   printf("\nEl resultado será: %d", sumatorio);
 }
```

→

Seguidamente presentaremos la simulación de este programa para entender mejor su ejecución y ver com ovan cambiando los valores de las distintas variables. Supondremos que el valor que el usuario introduce es –7 y como no es correcto, le pedirá que vuelva a introducir otro valor y supondremos que está vez el usuario introducirá un 3.

- int n, i, sumatorio;

?	?	?
n	i	sumatorio

Pantalla

- `printf("\nTeclee un valor entero positivo (>= 1): ");`

?	?	?
n	i	sumatorio

Pantalla

```
Teclee un valor entero positivo (>= 1): :
```

- scanf("%d", &n);

-7	?	?
n	i	sumatorio

Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
```

- n < 1? SÍ

-7	?	?
n	i	sumatorio

Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
```

- printf("\nEl valor tecleado no es valido.");

-7	?	?
n	i	sumatorio

Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
El valor tecleado no es valido
```

- printf("\nTeclee un valor entero positivo (>= 1): ");

-7	?	?
n	i	sumatorio

Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
El valor tecleado no es valido
Teclee un valor entero positivo (>= 1)::
```

- scanf("%d", &n);

3	?	?
n	i	sumatorio

Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
El valor tecleado no es valido
Teclee un valor entero positivo (>= 1):: 3
```

- n < 1? NO con lo cual este while acaba

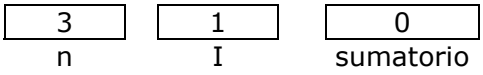
3	?	?
n	I	sumatorio

Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
El valor tecleado no es valido
Teclee un valor entero positivo (>= 1):: 3
```

- i = 1;

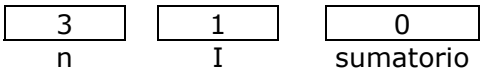
sumatorio = 0;



Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
El valor tecleado no es valido
Teclee un valor entero positivo (>= 1):: 3
```

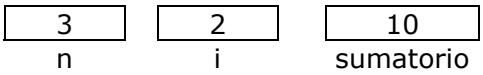
- i <= n ? SÍ



Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
El valor tecleado no es valido
Teclee un valor entero positivo (>= 1):: 3
```

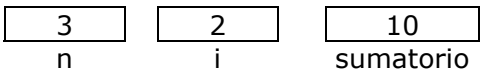
- sumatorio = sumatorio + ((8 * i) + 2); (el valor de i para esta asignación es 2)
  i = i + 1;



Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
El valor tecleado no es valido
Teclee un valor entero positivo (>= 1):: 3
```

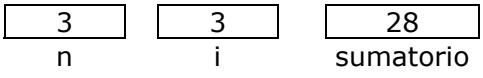
- i <= n ? SÍ



Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
El valor tecleado no es valido
Teclee un valor entero positivo (>= 1):: 3
```

- sumatorio = sumatorio + ((8 * i) + 2)); (el valor de i para esta asignación es 2)
  i = i + 1;



Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
El valor tecleado no es valido
Teclee un valor entero positivo (>= 1):: 3
```

- i <= n ? SÍ

3	3	28
n	i	sumatorio

Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
El valor tecleado no es valido
Teclee un valor entero positivo (>= 1):: 3
```

- sumatorio = sumatorio + ((8 * i) + 2)); (el valor de i para esta asignación es 3)
  i = i + 1;

3	4	54
n	i	sumatorio

Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
El valor tecleado no es valido
Teclee un valor entero positivo (>= 1):: 3
```

- i <= n ? NO, así que el while se acaba

3	4	54
n	i	sumatorio

Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
El valor tecleado no es valido
Teclee un valor entero positivo (>= 1):: 3
```

- printf("\nEl resultado es: %d", sumatorio);

3	4	54
n	i	sumatorio

Pantalla

```
Teclee un valor entero positivo (>= 1): : -7
El valor tecleado no es valido
Teclee un valor entero positivo (>= 1):: 3
El resultado es: 54
```

3.2 Un while dentro de otro while

Dentro de un while podría aparecer cualquier combinación de instrucciones. En los ejemplos que hemos visto hasta ahora solo nos han aparecido asignaciones, condicionales, instrucciones de entrada y salida y operaciones aritméticas. Ahora vamos a presentar un ejemplo en el que aparece un while dentro de otro.

3.2.1 Dentro de un while podemos encontrar otro (o varios) while

Este ejercicio consiste en escribir por pantalla la siguiente configuación de números:

```
1 2 3 4
1 2 3 4
1 2 3 4
```

Como se puede apreciar aparecen escritas 3 líneas donde por cada línea aparece una serie ascendente de números. Pues bien, para ir generando las líneas utilizaremos un while (el while_1). Y por cada vuelta de ese while (el while_1, es decir dentro del while_1) utilizaremos otro while (while_2) para generar la secuencia ascendente de números.

```
#include <stdio.h>

void main()
 {
   int i, j;

   i = 1; /* Porque estamos en la primera línea. */

   while (i <= 3)/* este será el while_1*/
    {
      j = 1; /* Por cada línea y como tenemos que generar una secuencia
.....ascendente de números que comienza con 1 utilizaremos j como
.....variable que contendrá los valores sucesivos desde 1 a 4.
               Por eso hay que inicializarlo a 1. ¡CUIDADO! j se iniciliza
 .....aquí y no  al comienzo por que POR CADA LÍNEA DEBEREMOS COMENZAR
               DE NUEVO LA SERIE ASCENDENTE*/
      while (j <= 4)
       {
         printf("%d  ", j);
         j = j + 1;   /*así j irá tomando el siguiente valor */
       }
      i = i + 1; /* llegaremos aquí cada vez que hayamos finalizado de
                    escribir una serie ascendente, y entonces deberemos pasar a
                    la siguiente linea avanzando con i que nos marcará en que
                    línea estamos. */
      printf("\n"); /* Esta instrucción nos permite generar en la pantalla
                       un salto de línea. */
    }

 }
```

Seguidamente presentaremos la simulación de este programa para entender mejor su ejecución y ver como van cambiando los valores de las distintas variables:

- int i, j;

?	?
i	j

Pantalla

- i = 1;
  i <= 3? SÍ

1	?
i	j

Pantalla

❖ j = 1;
j <= 4 ? SÍ

1	1
i	j

Pantalla

❖ printf("%d ", j); (une honetan j-ren balioa 1 da)
j = j + 1;

1	2
i	j

Pantalla

```
1
```

❖ j <= 4 ? SÍ (en este momento j vale 2)

printf("%d ", j);
j = j + 1;

1	3
i	j

Pantalla

```
1  2
```

→

❖ j <= 4 ? SÍ (en este momento j vale 3)

printf("%d ", j);
j = j + 1;

1	4
i	j

Pantalla

```
1  2  3
```

❖ j <= 4 ? SÍ (en este momento j vale 4)

printf("%d ", j);
j = j + 1;

1	5

i	j

Pantalla

```
1 2 3 4
```

- j <= 4 ? NO (en este momento j vale 5 así que no entraremos en el while_2, pero recordemos que la condición del while_1 puede que todavía sigua siendo cierta, aun queda comprobarlo)

1	5
i	j

Pantalla

```
1 2 3 4
```

- i = i + 1; (aumentamos i para indicar que hemos inpreso una nueva línea)
  printf("\n"); (con esta instrucción saltamos en la pantalla a la siguiente línea)

2	5
i	j

Pantalla

```
1 2 3 4
```

- i <= 3? SI (la condición el while_1, el exterior, todavía sigue siendo cierta)

2	5
i	j

Pantalla

```
1 2 3 4
```

- j = 1;
  j <= 4? SÍ

2	1
i	j

Pantalla

```
1 2 3 4
```

- printf("%d ", j); (ahora mismo j vale 1)
  j = j + 1;

2	2
i	J

Pantalla

```
1  2  3  4
1
```

❖ j <= 4? SÍ (ahora mismo j vale 2)
printf("%d ", j);
j = j + 1;

2	3
i	J

Pantalla

```
1  2  3  4
1  2
```

❖ j <= 4? SÍ (ahora mismo j vale 3)
printf("%d ", j);
j = j + 1;

2	4
i	J

Pantalla

```
1  2  3  4
1  2  3
```

❖ j <= 4? SÍ (ahora mismo j vale 4)
printf("%d ", j);
j = j + 1;

2	5
i	j

Pantalla

```
1  2  3  4
1  2  3  4
```

- j <= 4 ? NO (en este momento j vale 5 así que no entraremos en el while_2, pero recordemos que la condición del while_1 puede que todavía sigua siendo cierta, aun queda comprobarlo)

2	5
i	j

Pantalla

```
1 2 3 4
1 2 3 4
```

- (aumentamos i para indicar que hemos inpreso una nueva línea)
  printf("\n"); (con esta instrucción saltamos en la pantalla a la siguiente línea)

3	5
i	j

Pantalla

```
1 2 3 4
1 2 3 4
```

- i <= 3? SÍ (así va a comenzar otra vuelta del while_1, es decir del while experior)

3	5
i	j

Pantalla

```
1 2 3 4
1 2 3 4
```

- j = 1;
  j <= 4? SÍ

3	1
i	j

Pantalla

```
1 2 3 4
1 2 3 4
```

- printf("%d ", j); (j valdrá1)
  j = j + 1;

3	2
i	j

Pantalla

```
1 2 3 4
1 2 3 4
1
```

- j <= 4? SÍ (en este punto j vale 2)
  printf("%d ", j);
  j = j + 1;

3	3
i	J

Pantalla

```
1 2 3 4
1 2 3 4
1 2
```

- j <= 4? SÍ (en este punto j vale 3)
  printf("%d ", j);
  j = j + 1;

3	4
i	J

Pantalla

```
1 2 3 4
1 2 3 4
1 2 3
```

- j <= 4? SÍ (en este punto j vale 4)
  printf("%d ", j);
  j = j + 1;

3	5
i	j

Pantalla

```
1 2 3 4
1 2 3 4
1 2 3 4
```

- j <= 4 ? NO (en este momento j vale 5 así que no entraremos en el while_2, pero recordemos que la condición del while_1 puede que todavía sigua siendo cierta, aun queda comprobarlo)

3	5
i	j

Pantalla

```
1 2 3 4
1 2 3 4
1 2 3 4
```

i = i + 1; (aumentamos i)
printf("\n"); (En pantalla saltamos a la siguiente línea)

4	5
i	j

Pantalla

```
1 2 3 4
1 2 3 4
1 2 3 4
```

→

i <= 3? NO (Se acaba el while_1, el while exterior)

4	5
i	j

Pantalla

```
1  2  3  4
1  2  3  4
1  2  3  4
Pulsar una tecla para terminar.
```

Habremos entendido este ejercicio si somos conscientes de que por cada valor que toma i (es decir mientras i no cambia su valor) j va tomando valores desde 1 a 5 (el 5 no se escribirá porque no cumplirá <=4). En la siguiente tabla se ve claramente las diferentes combinaciones de valores que van tomando i y j.:

i	j
?	?
1	?
1	1
1	2
1	3
1	4
1	5
2	5
2	1
2	2
2	3
2	4
2	5
3	5
3	1
3	2
3	3
3	4
3	5
4	5

3.2.2 Dentro de un while podremos encontrar también dos whiles independientes

Este ejercicio consiste en pedirle un valor al usuario (n>=1) y escribir por pantalla a partir de n la siguiente configuación de números :

```
1
1  2  1
1  2  3  2  1
...
1  2  3  4  5  6  7  8  9  10 ...  n  ...10  9  8  7  6  5  4  3  2  1 (habrá n líneas)
←núm. en orden ascendente →   ←núm. en orden descendente→
```

Para generar las líneas de 1 a n necesitaremos un while, y por cada vuleta de este while exterior, necesitaremos 2 while independientes, uno para crear la secuencia ascente y otro para generar la secuencia descendente.

La petición de datos deberá repetrse hasta que el valor de n sea correcto:

```
#include <stdio.h>

void main()
 {
   int n, i, j;

  printf("\nTeclee un valor entero positivo (>= 1): ");
  scanf("%d", &n);

   while (n < 1)
    {
      printf("\nEl valor tecleado no es valido.");
      printf("\nTeclee un valor entero positivo (>= 1): ");
      scanf("%d", &n);
    }

   i = 1;

   while (i <= n) /*while_1*/
    {
   /*ahora deberemos utilizar otro while para generar por cada línea la
   secuencia ascendente de números*/
      j = 1; /* i lerroko lehenengo balioa */
      while (j <= i)
       {
         printf("%d  ", j);
         j = j + 1; /* i lerroan hurrengo balioa hartzeko */
       }
    /* al salir del while_2 j valdrá i+1*/
   /*llegados a este punto parte de la línea estará escrita. La parte
   correspondiente a la secuencias ascendente. Ahora deberemos utilizar
   otro while para generar el resto de la línea, es decir, la
   secuencia ascendente de números*/

      j = j - 2; /* al hacer esto y como j valía (i + 1) ahora valdrá i-1. */

      while (j >= 1)
       {
         printf("%d  ", j);
         j = j - 1; /*Vamos generando la secuancia descendente*/
       }

      i = i + 1; /* incrementamos el contador de líneas. */
      printf("\n"); /* En pantalla saltamos de línea. */
    }
 }
```

En esta solución se nos presentan dos whiles independientes dentro de un gran while. Es importante darse cuenta de que SOLO pasaremos a la siguiente vuelta del GRAN while (while_1) cuando hayamos salido de los whiles internos. Al entrar en una nueva vuelta del GRAN while (while_1) se volverán a ejecutar los whiles internos y así sucesivamente hasta que NOS SALGAMOS del GRAN while (while_1).

Otro ejemplo de un while anidado dentro de otro

Sumatorio doble

En este ejercicio se le pedirá al usuario dos valores para n y m tal que (n>=0) y (m>=0). Así una vez obtenidos los valores se calculará el siguiente sumatorio doble.:

$$\sum_{i=0}^{n}\sum_{j=0}^{m}(i*j)$$

El proceso de petición de valores para n y m se hará conjuntamente y se repetirá tantas veces como sea necesario hasta que AMBOS sean correctos:

```
#include <stdio.h>

/* El resultado será: El valor que se obtenga tras hacer las siguientes sumas

             (0 * 0) + (0 * 1) + (0 * 2) + ... + (0 * m) +
             + (1 * 0) + (1 * 1) + (1 * 2) + ... + (1 * m) +
             + (2 * 0) + (2 * 1) + (2 * 2) + ... + (2 * m) +
             + ... +
             + (n * 0) + (n * 1) + (n * 2) + ... + (n * m) +
*/

void main()
 {
   int m, n, sumatorio, i, j;

   printf("\nIntroduce dos valores positivos separados por comas ");
   scanf("%d, %d", &n, &m);

   while ((n < 0) || (m < 0))
    {
      printf("\n¡¡Alguno de los valores es incorrecto!!.");
      printf("\nIntroduce dos valores positivos separados por comas ");
      scanf("%d, %d", &n, &m);
    }

   sumatorio = 0;
   i = 0;

   while (??????????????????????????)
   {
    j = ????;

    while (????????????????????)
    {
     sumatorio = sumatorio + (i * j);
     j = ????????;
     }
     i =?????????????;
     printf("Para n=%d y m=%d el valor del sumatorio es %d", n, m, sumatorio);
    }

 }
```

4 EL DO-WHILE

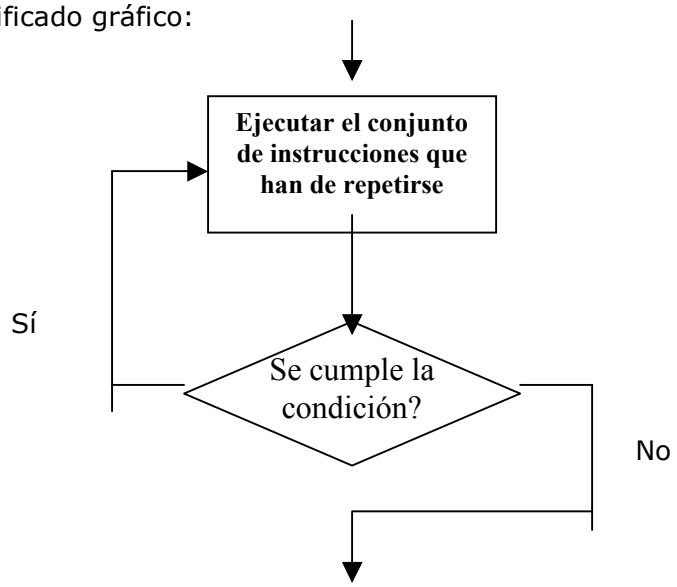
- Su forma:

```
do
  {
    Conjunto de instrucciones que han de repetirse
  }
while (condición);
```

- Su significado:

Las instrucciones que aparecen entre llaves se ejecutarán y luego se comprobará si la condición es cierta, así **Mientras** la condición siga siendo cierta se volverán a ejecutar las instrucciones que aparecen entre llaves.
EN TODO CASO LAS INSTRUCCIONES ENTRE LLAVES SE EJECUTARÁN **AL MENOS UNA VEZ**, la primera hasta comprobar si la condición es cierta.

- Su significado gráfico:



- do-while en castellano:

ejecutar {
Conjunto de instrucciones que han de repetirse
}**mientras**(condición) **sea cierta;**

- Su uso:

Resulta apropiado utilizar el Do-while cuando hemos de repetir la ejecuación de ciertas instrucciones y además sabemos que **POR LO MENOS LAS INSTRUCCIONES HAN DE EJECUTARSE UNA VEZ.** Por ejemplo, el proceso de petición de datos que ha de repetirse hasta que los valores introducidos sean correctos y sabemos que **COMO MÍNIMO HAREMOS LA PETICIÓN AL MENOS UNA VEZ.**

4.1 do-while y while:

Comparemos dos procesos de petición de datos uno implementado con un while el otro con un do-while. La petición de datos consistirá en pedir un valor tal que se encuantre entre el 1 y el 10 (amos inclusive, [1..10].

Con un while

```
#include <stdio.h>
void main(){
int num;

printf("Introduce un valor entre [1..10] ");
scanf("%d",&num);
while((num <1)||(num>10)){
……printf("Introduce un valor entre [1..10] ");
……scanf("%d",&num);
}
```

Con un Do-while

```
#include <stdio.h>
void main(){
int num;

do{
       printf("Introduce un valor entre [1..10] ");
       scanf("%d",&num);
   }while((num<1)||(num >10));
}
```

Cualquier cosa que se pueda expresar con un Do-while, prodrá a su vez expresarse con u while. El while es la más genérica de las intrucciones iterativas. Toda iteración se puede representar a través de un while, cosa que no es cierta para el Do-while. En los casos en los que no sepamos seguro que las instrucciones entre llaves se ejecutarán al menos una vez NO PODREMOS UTILIZAR EL Do-while.

Con el while podemos representar iteraciones que en algunos casos no lleguen a ejecutarse nunca. Por ejemplo recordemos el ejemplo 4.2.3.3, e cálculo de la media. Recordemos que se nos pedía calcular la nota media de los alumnos de una clase. Estableciamos un código con el profesor tal que si cuando éste introdujese un –1 como nota, eso nos indicaba que no quería introducir más notas.

```
#include <stdio.h>

void main()
 {
 float nota,suma_acumulada, numero_de_alumnos, media;

   suma_acumulada= 0; /* al comienzo y hasta no introducir ninguna nota
……..no habrá nada acumulado por lo tanto el valor será 0*/
   numero_de_alumnos= 0; /* Lo mismo con respecto al número de alumnos*/
    printf ("Introduce una nota por favor: ");
    scanf("%f",&nota);
     while (nota != -1){
        suma_acumulada= suma_acumulada+ nota;
        numero_de_alumnos = numero_de_alumnos+1;
        printf ("Introduce una nota por favor: ");
        scanf("%f",&nota);
```

```
    }
   if (numero_de_alumnos !=0){
              media= suma_acumulada/numero_de_alumnos;
   }
   else{
        media = 0;
        }
}
```

Supogamos que pudiese darse el caso de que el profesor no tuviera alumnos porque no se le haya matriculado ninguno. En ese caso es posible que las instrucciones entre llaves no lleguen a ejecutarse porque desde el comienzo elprofesor introduzca un –1. Este caso no podría implementarse con un Do-while puesto que las instrucciones entre llaves puede que no se ejecuten ni una sola vez. **Este sería un caso en el que se puede utilizar un while y no sería posible utilizar un Do-while.**

El siguiente caso es justamente el contrario. **En el siguiente caso se podrán utilizar ambos, pero sería recomendable utilizar un Do-while**:

Escribir un programa que vaya imprimiendo los número pares empezando por el 2; Por cada número que se imprima se le preguntará al usuario si quiere continuar. Si dijese que si (s) continuaríamos escribiendo el siguiente par, así hasta que deje de respnder que sí (s).
While

```
#include <stdio.h>
void main(){
int num;
char respuesta;

num=2;
printf("%d es par\n", num);
printf ("¿Quieres ver el siguiente número par?(s/n) ");
fflush(stdin);
scanf ("%c",&respuesta);

while(respuesta=='b'){
       num=num+2;
       printf("%d es par\n", num);
       printf ("¿Quieres ver el siguiente número par?(s/n) ");
       fflush(stdin);
       scanf ("%c",&respuesta);
}
```

Do-while

```
#include <stdio.h>
void main(){
int num;
char respuesta;

num=2;
do{
printf("%d es par\n", num);
printf ("¿Quieres ver el siguiente número par?(s/n) ");fflush(stdin);
scanf ("%c",&respuesta);
}while(respuesta=='b');
```

5 EL FOR

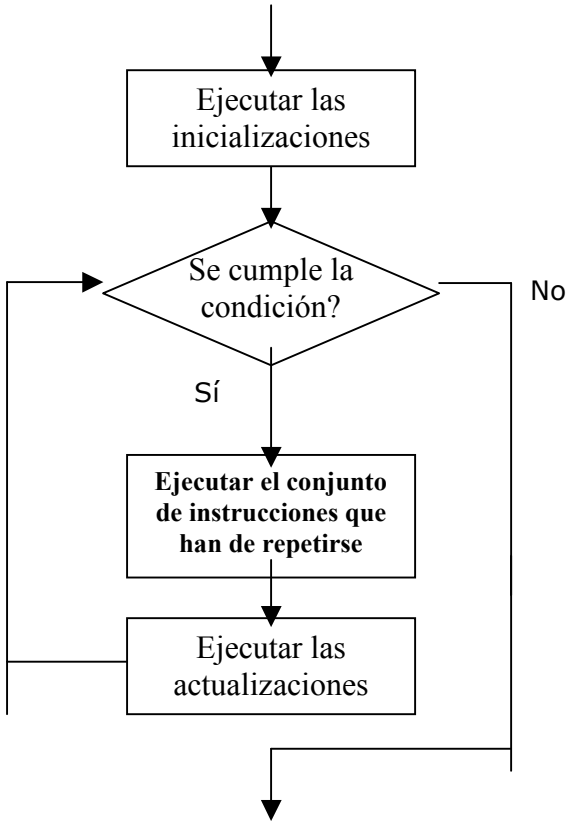
- Su forma:

```
for (inicialización; condición; actualización)
  {
   Conjunto de instrucciones que han de repetirse
  }
```

- Su significado:

Para empezar se ejecuta la inicialización (solo se ejecutará una vez al comienzo, no se repetirá). Acto seguido se comproará si la condición es cierta, y mientras sea cierta se ejecutarán tanto las instrucciones dentro de las llaves como las actualizaciones hasta que la condición deje de ser cierta.

- Su significado gráfico:



5.1 for y while:

En C el for y el while se pueden considerar como equivalentes. Pero en otros lenguajes no es el caso, así que nosotros vamos a limitar el uso del for a ciertos casos concretos (casos para los cuales en otros lenguajes también se prodría utilizar un for).

La traducción de un for a un while es directa

```
for (inicialización; condición; actualización)
  {
   Conjunto de instrucciones que han de repetirse
  }
```

```
inicialización
while (condición)
  {
    Conjunto de instrucciones que han de repetirse
    actualización
  }
```

¡¡¡PERO CUIDADO!!! **Nosotros solo vamos a utilizar el** for[2] cuando sepamos con certeza CUANTAS VUELTAS VAMOS A DAR. Es decir existe alguna variable que nos marque cuantas iteraciones se van a ejecutar.

Por ejemplo si nos dijesen que tenemos que ir pidiendo números al usuario hasta que éste introduzca un 0. ¿Podemos saber en que vuelta va a introducir el usuario el 0?. Es decir ¿sabemos cuantas veces hemos de realizar la iteración de la instrucción pedir un valor al usuario?. No, no lo podemos saber. Así que **NO UTILIZARÍAMOS UN FOR SINO UN WHILE.**

Si por el contrario nos dijesen que le pidamos al usuario 50 números. Aquí si sabemos cuantas veces vamos a repetir la instrucción de pedir un valor al usuario, de hecho 50 veces. **Así que aquí sería apropiado utilizar un for (también lo sería utilizar un while).**

Ahora vamos a resolver a través del uso de fors un ejercico ya resuel to whiles. Es un ejercicio en el que utilizabamos un while dentro de otro.

Deberemos escribir por pantalla la siguiente configuración de números:

```
1 2 3 4
1 2 3 4
1 2 3 4
```

[2] La instrucción for es muy particular en C :

Se pueden escribir fors que contengan inicializaciones y actualizaciones vacias. También fors que tengan la parte de la condición vacia; en este caso el for nos llevará a ejecutar un bucle infinito. Tanto en la parte de inicializaciones como en la de actualizaciones es posible utilizar scanf y printf. Además es posible escribir más de una inicialización y también más de una actualización.

Con whiles

```
#include <stdio.h>

void main()
 {
   int i, j;

   i = 1; /* Porque estamos en la primera línea. */

   while (i <= 3)/* este será el while_1*/
    {
      j = 1; /* Por cada línea y como tenemos que generar una secuencia
.....ascendente de números que comienza con 1 utilizaremos j como
.....variable que contendrá los valores sucesivos desde 1 a 4.
               Por eso hay que inicializarlo a 1. ¡CUIDADO! j se iniciliza
 .....aquí y no  al comienzo por que POR CADA LÍNEA DEBEREMOS COMENZAR
               DE NUEVO LA SERIE ASCENDENTE*/
      while (j <= 4)
       {
         printf("%d  ", j);
         j = j + 1;   /*así j irá tomando el siguiente valor */
       }
      i = i + 1; /* llegaremos aquí cada vez que hayamos finalizado de
                    escribir una serie ascendente, y entonces deberemos pasar a
                    la siguiente linea avanzando con i que nos marcará en que
                    línea estamos. */
      printf("\n"); /* Esta instrucción nos permite generar en la pantalla
                       un salto de línea. */
    }

 }
```

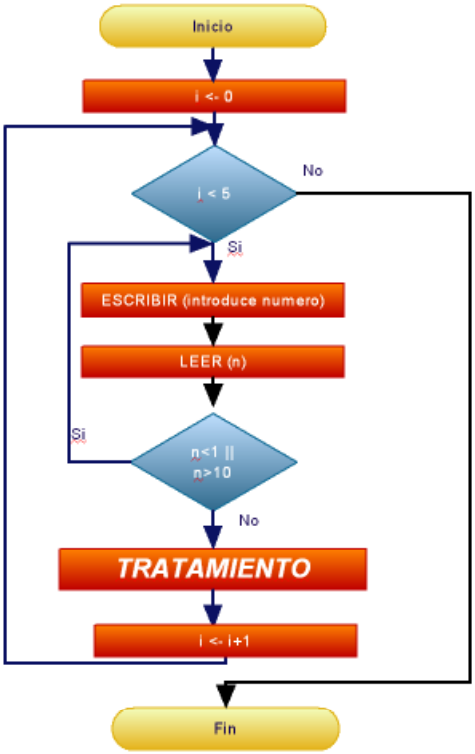
Con fors

```
#include <stdio.h>

void main()
 {
   int i, j;

   for (i = 1; i <= 3; i=i+1)
    {
       for (j = 1; j <= 4; j=j+1)
       {
         printf("%d  ", j);
       }
      printf("\n"); /* Esta instrucción nos permite generar en la pantalla
                       un salto de línea. */
    }
 }
```

6 ANEXO 1 VARIOS DIAGRAMAS DE FLUJO

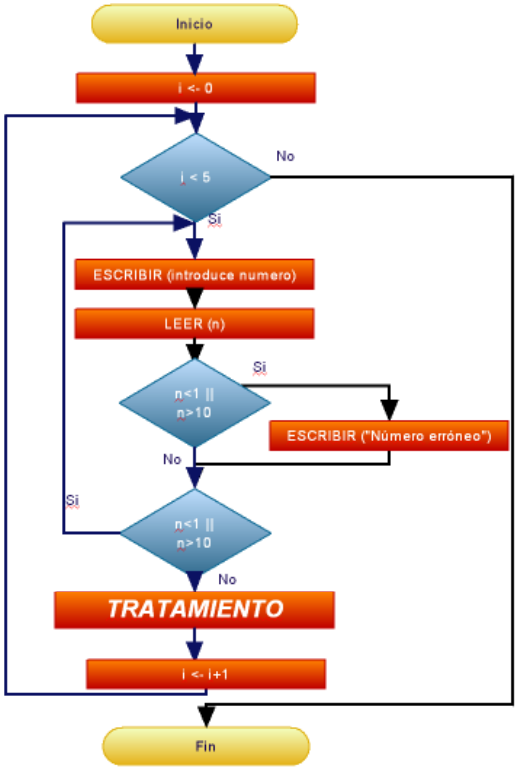
1-Pedir un número y validar que esté entre 1 y 10 y luego tratar.

2.Pedir 5 números validos que estén entre 1 y 10 tratando cada uno

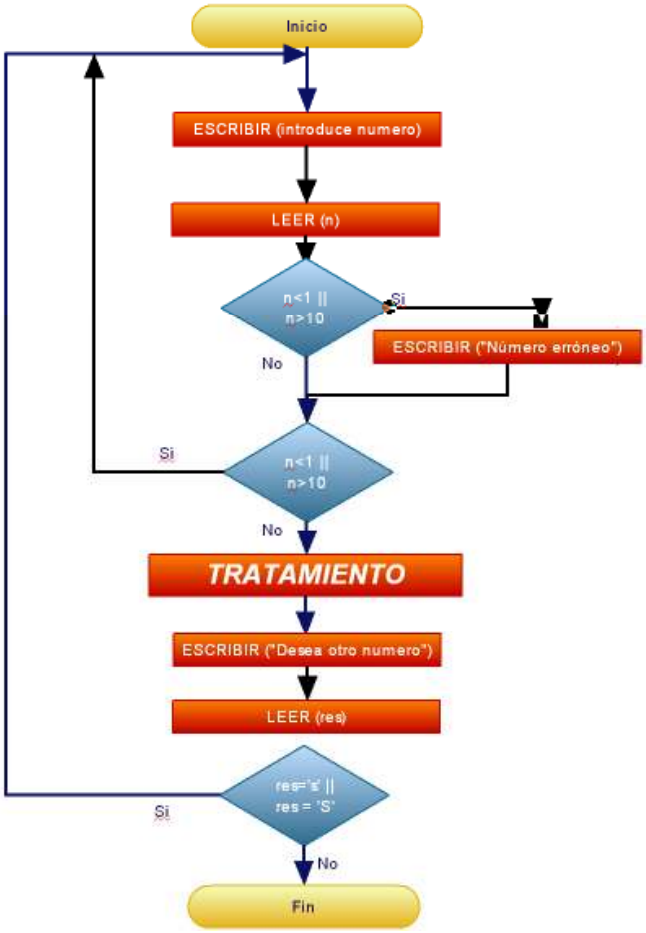


3.- Pedir números validos que estén entre 1 y 10 tratando cada uno y finalizando cuando el valor sea no válido

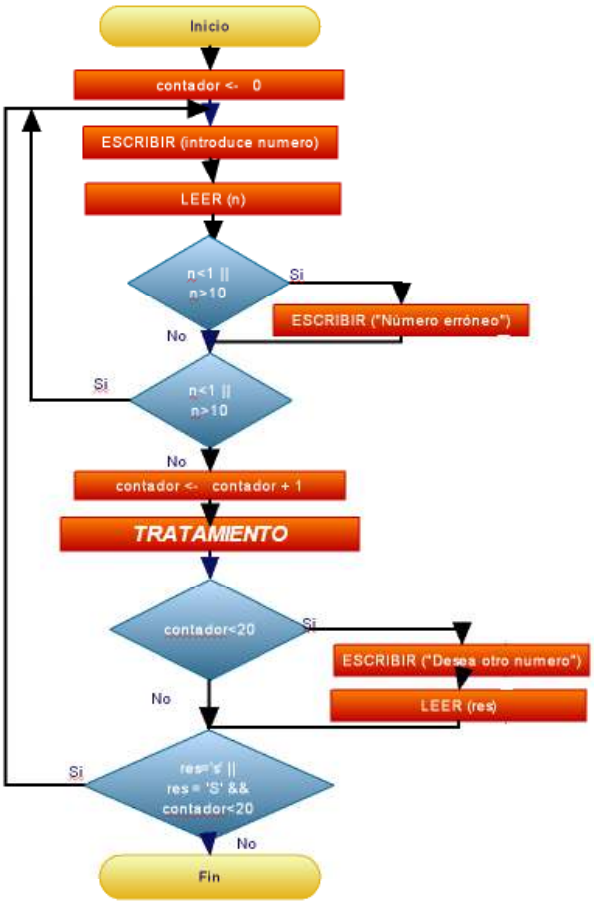
4.-Pedir 5 números validos que estén entre 1 y 10 tratando cada uno e incluyendo mensaje error cuando el valor no es válido



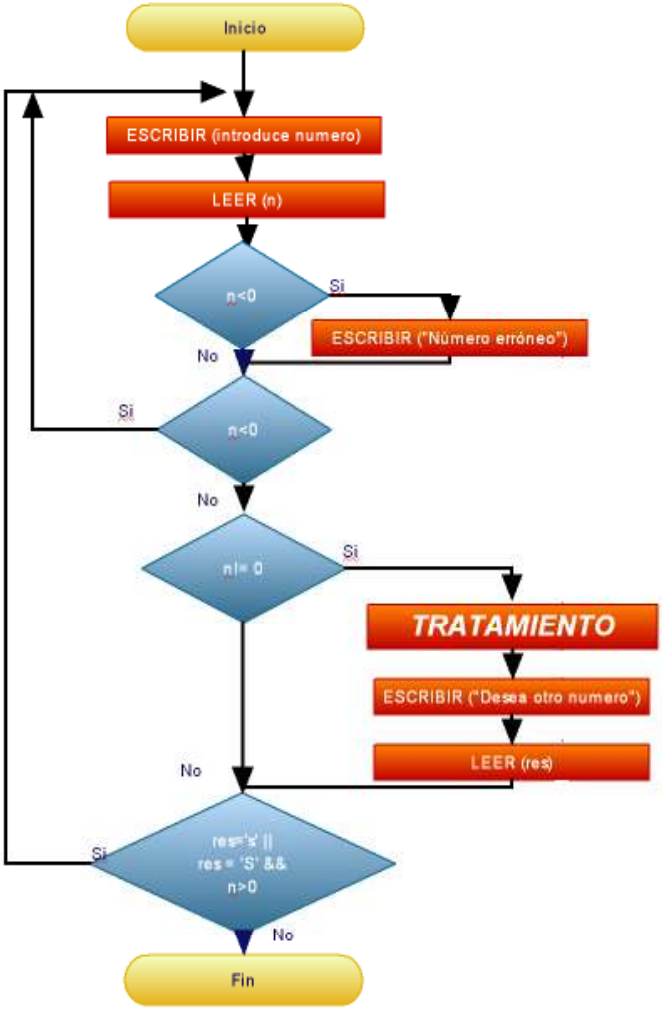
5.-Pedir números validos que estén entre 1 y 10 tratando cada uno e incluyendo mensaje error cuando el valor no es válido. Repetir hasta que el usuario decida no introducir más números con pregunta (s/n).



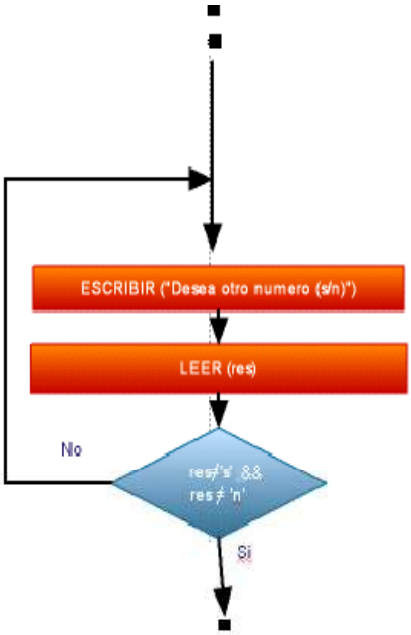
6.-Pedir números validos que estén entre 1 y 10 tratando cada uno e incluyendo mensaje error cuando el valor no es válido. Repetir hasta que el usuario decida no introducir más números con pregunta (s/n) o hasta que haya introducido 20 válidos.



7.- Pedir números validos que sean positivos tratando cada uno e incluyendo mensaje error cuando el valor no es válido. El 0 indica fin de datos y no se trata. Repetir hasta que el usuario decida no introducir más números con pregunta (s/n) o que haya introducido un 0.



8.- Solicitar respuesta del usuario hasta que sea ‘s’ o ‘n’ .
(A Añadir en cualquiera de las anteriores)



7 ANEXO 2 EJERCICIOS PARA HACER

1.- Mostrar los números del 1 al 11 con do_while y while
2.- Mostrar los múltiplos de 6 del 0 hasta el 10.000
3.-Imprimir los números del 1 al 10 5 veces
4.-Mostrar la tabla de multiplicar de los números del 1 al 9
5.-Mientras no adivine un número al azar entre 0 y 99 número indicar < o >
6.-Calcular el sumatorio de los números introducidos hasta 0 o 10 números.
7.- Lo mismo pero el productorio
7- Pedir notas para alumnos y mostrar la media .
    a) Hasta que la nota sea 0
    b) 10 alumnos
8.-Suma de pares e impares hasta el número introducido por el usuario
9.- Escribe y prueba en el ordenador un programa que lea una secuencia de números enteros introducidos por el usuario a través del teclado. El programa irá sumando los

valores a medida que se introducen, hasta que la suma supere un valor establecido previamente mediante una constante del programa. Cuando eso sucede el programa escribe en pantalla el valor de la suma y acaba.
10.- De 1 hasta el limite introducido por el usuario sacar en dos columnas num y num al cuadrado.
11.- Sacar la tabla de códigos ASCII de las minusculas y las mayúsculas
12.-Obtener la tabla de precios por gramos de 5 hasta 60 gramos. (5 gramos=20 euros y cada 5 gramos más son 12 euros adicionales).

5 gramos 20 euros
10 gramos 32 euros
....

13.-Introducir caracteres hasta que se introduzca el carácter punto y contar los blancos indicando tras la lectura del punto el número de blancos introducido.
14.-Dada una secuencia de caracteres finalizada con un punto, calcular cuántas veces aparece la letra 'A'.
15.-Dada una secuencia de caracteres finalizada con un punto, calcular cuántas vocales contiene.
16.-Dada una secuencia de caracteres finalizada con un punto, calcular cuántas vocales, cuántas no vocales y el total de caracteres que contiene.