ESCUELA POLITÉCNICA SUPERIOR

INFORMÁTICA – CURSO 2024-25

PRÁCTICA 3. ESTRUCTURAS ITERATIVAS

OBJETIVOS

En esta práctica aprenderemos el uso de las estructuras iterativas `while` y `for`, las cuales sirven para repetir una o varias acciones mientras se cumpla una condición. También reforzaremos los conocimientos adquiridos en las prácticas anteriores mediante programas que combinen estas nuevas estructuras con ya estudiadas.

HASTA AHORA...

En prácticas anteriores se ha aprendido:

- La estructura principal de un programa en C: la función `main` y las librerías.
- Variables y Constantes: con sus respectivos tipos, usos y funcionamiento.
- Funciones `printf/scanf`: muestra y obtención de información por pantalla/teclado.
- Condicionales `if/else`: ejecución de un programa en árbol.

PRERREQUISITOS

Para el correcto desarrollo de esta práctica, el alumno ANTES DE LA SESIÓN DE PRÁCTICA EN EL LABORATORIO, debe leer con atención y comprender el contenido de la parte teórica de dicha práctica, el apartado ESTRUCTURAS ITERATIVAS de este documento. Además el alumno habrá trabajado previamente los tres primeros ejercicios descritos en el apartado EJERCICIOS de este documento, así como, los ejercicios de la práctica anterior.

Se aconseja el uso del material de apoyo de la asignatura. Así, para esta práctica el alumno puede consultar la siguiente bibliografía:

- Rodríguez Jódar, M.A. y otros, "Fundamentos de informática para ingeniería industrial":
    - Del *Capítulo 4, Fundamentos de programación*: leer los apartados siguientes:
        - **Apartado 4.4, *Estructuras iterativas***
        - **Apartado 4.4.1, *El bucle while***
        - **Apartado 4.4.3, *El bucle for***
        - Apartado 4.4.4, *Anidamiento de bucles*
- García de Jalón de la Fuente, J. y otros, "Aprenda lenguaje ANSI C como si estuviera en primero":
    - Del *Capítulo 5 Control del Flujo de Ejecución:* leer el apartado *5.2 Bucles*.

Introducción a las estructuras iterativas

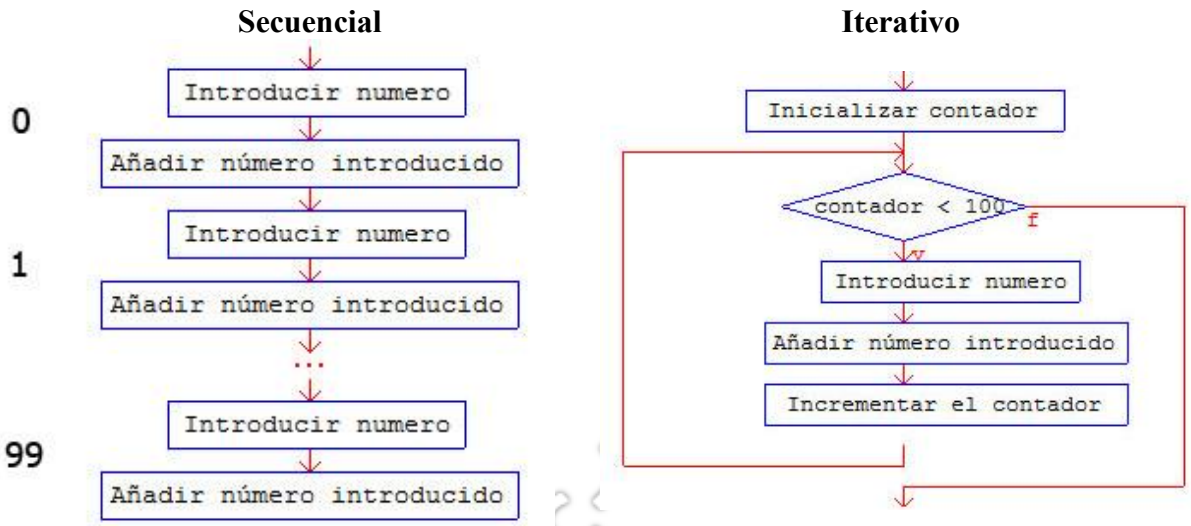
Uno de los usos para los que se crearon los ordenadores fue la realización de tareas repetitivas. Con lo visto hasta ahora la única forma que tenemos de ejecutar varias veces un código es copiarlo tantas veces como queramos que se repita:

```
main(){
        int a,b=0;
        printf("Voy a sumar 100 números\n");

        printf("Dame un número entero: ");
        scanf("%d", &a);
        b=b+a;
        printf("Dame un número entero: ");
        scanf("%d", &a);
        b=b+a;
        … /* hasta 100 veces… */
}
```

Sin embargo, los lenguajes de programación proporcionan una serie de estructuras que permiten repetir la ejecución de una porción de código. En el diagrama de flujo esta repetición se refleja mediante una flecha que asciende hasta el punto del programa en el que comienza la repetición.

Los siguientes diagramas representan parte del programa mostrado anteriormente en su versión secuencial y no iterativa. La versión secuencial da lugar a un programa muchísimo mayor que la versión iterativa. Por otro lado, la versión iterativa suele utilizar estructuras condicionales para controlar el número de repeticiones.



Las estructuras iterativas, además de evitar repetir el mismo código, resultan imprescindibles en muchas ocasiones ya que el número de repeticiones es variable según cómo el usuario interactúe con el programa.

ESTRUCTURA WHILE

La **estructura *while*** repite un bloque de código mientras se cumple una condición. En la figura anterior, el diagrama de flujo de la derecha muestra un ejemplo de estructura while. Esta estructura se representa ayudándonos de una estructura condicional la cual se encarga de salir del bucle cuando la condición deja de cumplirse. Como podemos apreciar en dicho diagrama, la rama verdadera de la condición ejecuta el bloque de instrucciones a repetir y retorna nuevamente a evaluar la condición, en cambio la rama falsa de la condición continua más allá del bucle dando por terminada la ejecución del bucle.

En lenguaje C, la sintaxis de **`while`** es la siguiente:

```
while (condición) {
        sentencia1;  // Instrucciones que se van a repetir
        sentencia2;
        ...
}
```

Las sentencias incluidas entre las llaves corresponden al bloque de instrucción que se ejecutarán **mientras** se cumpla la condición. Esta condición se expresa empleando los mismos operadores lógicos y relacionales que estudiamos para la estructura *if-else.*

La ejecución de una estructura `while` se detalla en el siguiente algoritmo:

1. *Se evaluaría la condición de la cabecera del* `while`*.*

- *a. Si la condición es falsa, salta al final de la llave que cierra el* `while` *y continúa la ejecución del resto de instrucciones del programa.*
- *b. Si la condición es verdadera, ejecuta TODAS las instrucciones contenidas en el cuerpo de la estructura* `while` *en orden secuencial hasta llegar a la llave de cierre del* `while`*.*

2. *Se salta de vuelta a la cabecera del* `while` *y, por lo tanto, al* ***punto 1*** *de este algoritmo.*

Pero ¿qué es lo que se indica como condición de la estructura *while*? Dependerá del programa que estemos implementando, pero, principalmente podrá ser de dos tipos:

- Número concreto de repeticiones: haciendo uso de una variable que actúa de contador para indicar el número de repeticiones realizadas (suele ser de tipo entero). Inicialmente se le asignará el valor 0 y la condición del bucle irá destinada a verificar su se ha alcanzado el número de iteraciones que se pretendía *while*.

  Así pues, si se quiere ejecutar 10 veces, la variable que hace de contador (supongamos que se llama "c") se iniciará con un 0 y la condición indicada en el *while* será "c<10".

  ¿Por qué menos que 10 y no menor o igual? Pues porque se ha iniciado en el 0, no en el 1, con lo cual del 0 al 9 (inclusive) ya se contabilizan diez repeticiones.

  Hay que tener en cuenta que la variable que actúa de contador debe ser incrementada dentro de las instrucciones del *while* (normalmente la última de ellas) para que la condición se modifique y no nos quedemos atrapados dentro de la estructura iterativa para siempre.

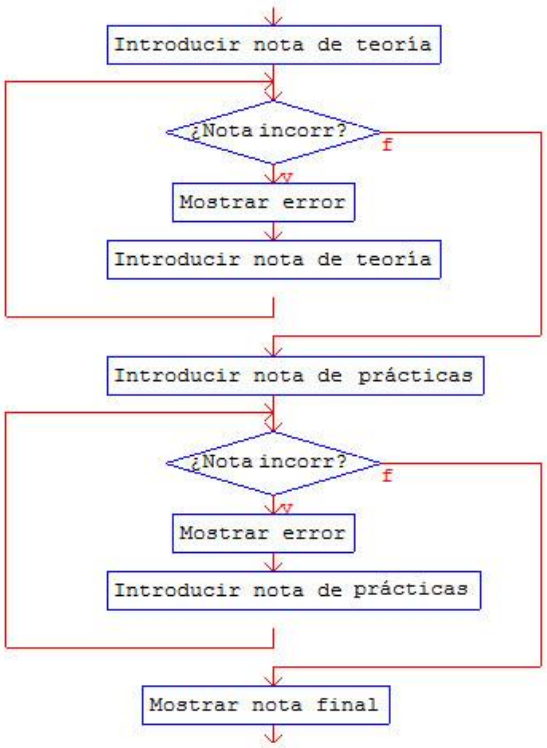
  Por ejemplo, el código anterior con una estructura *while* quedaría:

```
main(){
        int a, b=0, contador=0;
        printf("Voy a sumar 100 números\n");

        while(contador<100){
                printf("Dame un número entero: ");
                scanf("%d", &a)
                b=b+a;
                contador++;
        }
        printf("El resultado de la suma es %d\n", b);
        }
```

- Número indeterminado de repeticiones: búsqueda de un determinado valor o rango de valores en una o más variables. Dentro de esta categoría podrían englobarse multitud de diferentes tipologías de programas; es por ello más interesante ver un ejemplo concreto, ya que se podrán estudiar muchos casos mediante los ejercicios.

Ejemplo: introducción de un valor por teclado que debe encontrarse dentro de un rango determinado y, en caso de no estarlo, se volverá a pedir nuevamente.




```
main(){
    int notaTeoria, notaPractica;
    printf("Calcular nota final\n");

    printf("Introduce nota de teoría: ");
    scanf("%d", &notaTeoria);

    while(notaTeoria<0 || notaTeoria>10){
      printf("Error, vuelve a   introducirla ");
      scanf("%d", &notaTeoria)
    }
    printf("Introduce nota de práctica: ");
    scanf("%d", &notaPractica);

    while(notaPractica<0|| notaPractica>10){
      printf("Error, vuelve a introducirla: ");
      scanf("%d", &notaPractica)
    }

    printf("Nota:    %f\n",    notaTeoria*0.4    +
      notaPractica*0.6);
}
```

Este mismo caso se podría plantear sin estructuras iterativas pero si el usuario volviera a introducir el valor de teclado mal ya no se podría hacer nada al respecto. Es por ello que las estructuras iterativas son muy útiles para estas cuestiones.
Fíjese como en estos casos no se conoce a priori el número de repeticiones que se van a ejecutar de cada estructura iterativa, ya que depende de la "terquedad" del usuario. Igualmente, la condición se ve modificada directamente con la función `scanf`, así que en algún momento saldremos de la estructura iterativa.

Las estructuras iterativas también se llaman **bucles**, y a cada ejecución completa de las sentencias contenidas en el bucle se le denomina **iteración**.

¡IMPORTANTE! Tal como se indicó antes, en cada iteración del bucle debe haber alguna sentencia que deje de cumplir la condición del *`while`* para poder salir del bucle. De lo contrario tendríamos un **bucle infinito** y el programa no terminaría su ejecución nunca.

ESTRUCTURA "FOR"

La sentencia *`for`* es otra forma de repetir un mismo bloque de instrucciones varias veces. Su sintaxis es más compacta que la de la sentencia *`while`* pues permite escribir en una sola línea la condición de comprobación del bucle, la sentencia que nos actualiza el contador y la inicialización de dicho contador. Veamos su sintaxis:

```
for (inicialización; condición; actualización) {
      sentencia1; // Instrucciones que se van a repetir
      sentencia2;
      ...
}
```

El funcionamiento detallado paso a paso sería el siguiente:

1. *Se ejecutaría el código correspondiente a la* ***inicialización*** *del* for.
2. *Se evaluaría la* ***condición*** *de la cabecera del* for.
   a. *Si la condición es falsa, salta al final de la llave que cierra el* for *y prosigue la ejecución del resto de instrucciones del programa.*
   b. *Si la condición es verdadera, ejecuta TODAS las instrucciones contenidas en el cuerpo de la estructura* while *en orden secuencial hasta llegar a la llave de cierre del* while.
3. *Se salta de vuelta a la cabecera del* for.
4. *Se ejecutaría la* ***actualización*** *de la/s variable/s de control del bucle.*
5. *Se vuelve al* ***paso 2*** *de este algoritmo.*

Veamos un ejemplo para comprender su funcionamiento:

```
…
for(i=0; i<10; i++) {
      /* instrucciones internas */
}
…
```

Traza de ejecución:

"i"	Situación	¿Qué pasa antes?
0	Se comprueba que cumple la condición "i<10" y entra en el bucle.	1er acceso al *for*. Inicialización "i=0".
1	Se comprueba que cumple la condición "i<10" y entra en el bucle.	Incremento de "i".
2	Se comprueba que cumple la condición "i<10" y entra en el bucle.	Incremento de "i".
3	Se comprueba que cumple la condición "i<10" y entra en el bucle.	Incremento de "i".
4	Se comprueba que cumple la condición "i<10" y entra en el bucle.	Incremento de "i".
5	Se comprueba que cumple la condición "i<10" y entra en el bucle.	Incremento de "i".
6	Se comprueba que cumple la condición "i<10" y entra en el bucle.	Incremento de "i".
7	Se comprueba que cumple la condición "i<10" y entra en el bucle.	Incremento de "i".
8	Se comprueba que cumple la condición "i<10" y entra en el bucle.	Incremento de "i".
9	Se comprueba que cumple la condición "i<10" y entra en el bucle.	Incremento de "i".
10	Se comprueba que NO cumple la condición "i<10", sale del bucle y continua la ejecución del programa.	Incremento de "i".

Fíjese como, tras salir del bucle, el valor de "i" es 10, ya que se realizó un incremento previo a comprobar que no satisfacía la condición del bucle.

Resumiendo: Las sentencias incluidas entre las llaves se ejecutarán **mientras** se cumpla la condición**.** Además, en cada iteración del bucle, tras las sentencias incluidas entre las llaves, se ejecutará la sentencia de actualización.

Tal como se comentó anteriormente, tanto "for" y "while" son sentencias destinadas a repetir un conjunto de instrucciones durante un número concreto (determinado o indeterminado) de instrucciones. Entonces, ¿por qué existen las dos y no solo una de ellas? Pues por simplicidad en determinados aspectos.

Aun así, veamos dos casos de traducción de uno a otro:

Bucle WHILE	*Bucle FOR*
Caso 1: número conocido de repeticiones	

`int i;`
`i=0; //Inicialización`
`while(i<20) { //Condición`
`  printf("%d, ", i);`
`  i++; // Modificación`
`}`
	`int i;`
	`for(i=0; i<20; i++) {`
	`  printf("%d, ", i);`
	`}`

Caso 2: número no conocido de repeticiones	

`int nota;`
`printf("Introduzca la nota: ");`
`scanf("%d", &nota); //Inicialización`
`while(nota<0 || nota>10) { //Condición`
`  printf("Incorrecto. Pruebe otra vez: ");`
`  scanf("%d", &nota); // Modificación`
`}`
	`int nota;`
	`printf("Introduzca la nota: ");`
	`for(scanf("%d", &nota);nota<0 || nota>10;scanf("%d",&nota);){`
	`  printf("Incorrecto. Pruebe otra vez: ");`
	`}`

Como puede comprobar, la traducción es inmediata una vez que se han identificado los tres elementos fundamentales: inicialización, condición y modificación. Sin embargo, hay determinados casos en los que el código resultante no es muy comprensible o en los que resulta más cómodo y entendible realizarlos de otra forma.

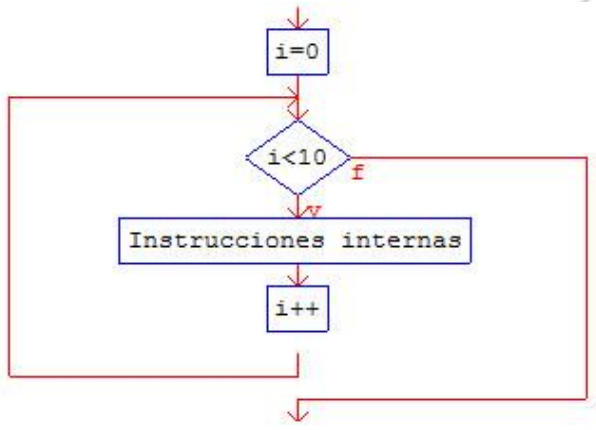
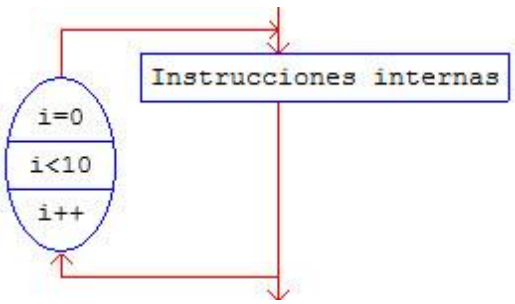
Así pues, lo más común es:

- ***Casos con un número conocido de repeticiones → bucle FOR***
- ***Casos con un número NO conocido de repeticiones → bucle WHILE***

Puesto que pueden emplearse indistintamente, el uso de uno u otro no es más que una recomendación por lo que puede utilizar el que resulte le resulte más cómodo.

Para representar el diagrama de flujo de un bucle `for` suele usarse la misma simbología que para `while` ya que ambas estructuras son equivalentes. No obstante ZinjaI emplea un símbolo propio para representar la estructura `for` que consiste en una elipse vertical divida en 3 partes para especificar la inicialización, la condición y la actualización del bucle.

Veamos el bucle for mediante un ejemplo utilizando ambos diagramas:

Bucle WHILE	*Bucle FOR*
	

EJERCICIOS:

1. Escribir un programa que:

- Muestre en pantalla todos los números desde el 1 hasta el 100 (inclusive).
- Muestre en pantalla todos los números desde el 100 al 1 (inclusive), en orden inverso.
- Muestre en pantalla todos los números pares que hay en los 100 primeros números naturales (100 inclusive).
- Muestre en pantalla todos los múltiplos de 3 que hay en los 100 primeros números naturales (100 inclusive).

Para hacer los dos últimos apartados, puede utiliza el operador módulo (%), el cual calcula el resto de una división. Por ejemplo, el resultado de *6%2* sería 0, ya que el resto de esta división es 0.

2. Complete el siguiente programa, en el cual se solicita al usuario que introduzca un número por teclado (que se almacenará en la variable *n*) y mientras este número no esté en el intervalo [0, 10] se sigue pidiendo un valor para *n*.

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

	printf(“Introduce un valor: ”);
	scanf(“%d”, &n);

	while(_________){

		printf(“Introduce otro valor: ”);
		____________

	}

	printf("El valor %d está dentro del rango", n);
}
```

Cambie el código anterior, de forma que la condición para que se siga introduciendo un valor por teclado sea que la variable *n* SÍ esté en el intervalo [0, 10], es decir, que se sigan pidiendo números por teclado hasta que dicho número esté fuera del intervalo.

3. Analice SIN utilizar ZinjaI el siguiente fragmento de código:

```
#include <stdio.h>
int main(){
	int x, y;
	x = 1;
	y = 2;

	while(x<3){

		if(x%2 == 0){
			y = y + 1;
			printf("A\n");
		}
		else{
			y = y + 2;
			printf("B\n");
		}

		printf("C\n");
		x++;
	}
	printf("fin: %d\n", y);
}
```

Escriba en la siguiente tabla los valores que van tomando las variables x e y:

x	y

¿Qué se mostraría por pantalla tras ejecutar el código?

4. Sumatorio. Supongamos que queremos sumar tres números que vamos a leer por el teclado. Se podría realizar lo siguiente (sin utilizar bucle):

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

	suma = 0;
	printf(“Introduce un valor: ”);
	scanf(“%d”, &n);
	suma = suma + n;

	printf(“Introduce un valor: ”);
	scanf(“%d”, &n);
	suma = suma + n;

	printf(“Introduce un valor: ”);
	scanf(“%d”, &n);
	suma = suma + n;

	printf(“La suma total es %d”, suma);

}
```

Hay tres líneas que se repiten continuamente, por lo que sería más conveniente realizarlo con un bucle

que ejecute tres veces este fragmento de código:

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

	suma = 0;

	for(i=0; i<3; i++){
		printf("Introduce un valor: ");
		scanf("%d", &n);
		suma = suma + n;
	}

	printf("La suma total es %d", suma);
}
```

Ejemplo de cómo funciona la variable acumuladora *suma* (análisis del código anterior dándole distintos valores a la variable *n*):

```
suma = 0
Inicialización bucle -> i = 0
Condición bucle -> ¿0<3? Sí -> 1ª Iteración:
"Introduce un valor" -> n = 3
suma = 0 + 3
Actualización bucle -> i++ -> i = 1

Condición bucle -> ¿1<3? Sí -> 2ª Iteración:
"Introduce un valor" -> n = 6
suma = (0+3) + 6 = 3 + 6 = 9
Actualización bucle -> i++ -> i = 2

Condición bucle -> ¿2<3? Sí -> 3ª Iteración:
"Introduce un valor" -> n = 2
suma = (0+3+6) + 2 = 9 + 2 = 11
Actualización bucle -> i++ -> i = 3

Condición bucle -> ¿3<3? No -> Fin bucle

Valor final de suma = 0 + 3 + 6 + 2 = 11 (acumula la suma de todos los valores de n)

"La suma total es 11"
```

Modifique el código anterior para que realice el sumatorio de **5** números recogidos por teclado.

5. Modifique el programa del ejercicio anterior para que, primero, el programa pida por teclado cuántos valores se van a sumar (guardándolo en una variable entera, llamada *numvalores,* la cual deberá declarar). Esto significa que la cantidad de números a sumar ya no es fija (como en los casos anteriores), sino que depende del valor introducido por teclado al principio. Tenga en cuenta que preguntar cuántos valores se van a sumar sólo se hace una vez.

6. Analice SIN utilizar ZinjaI el siguiente fragmento de código:

```
int x = 0;
int i;

for(i=1; i<5; i++){
       x = x + i;
}
printf("%d", x);
```

Escriba en la siguiente tabla los valores que van tomando las variables x e i:

x	i

¿Qué valor toma la variable x al finalizar el programa? ¿Cuál es su función?

7. Escriba un programa que calcule la suma de la sucesión de 1+2+3+…+n, donde n es un valor entero introducido por teclado. Es decir, el programa debe calcular la suma desde 1 hasta n incluido. Para ello, calcule la suma de dos formas:

- Mediante la siguiente fórmula: $\texttt{suma} = \frac{n(n+1)}{2}$
- Mediante una estructura iterativa

El resultado del sumatorio debe ser idéntico utilizando ambas formas.

8. Escriba un programa que calcule el producto de la sucesión 1×2×3×…×n, donde n es un valor entero introducido por teclado. Es decir, el programa debe calcular el producto desde 1 hasta n incluido. Esto se conoce como el factorial de un número. Tenga en cuenta que tendrá que acumular el **producto** de cada uno de estos valores, y no la suma como en los ejercicios anteriores.

9. Escriba un programa que calcule la potencia de un número real elevado a un exponente entero positivo. Los valores de la base (*b*) y el exponente (*e*) se leen de teclado. El cálculo de la potencia puede entenderse como el producto de la sucesión de *b*×*b*×*b*×…×*b*, tantas veces como indique *e*.

10. Escribir un programa que reciba por teclado números enteros hasta que el usuario introduzca el 0 y que calcule el sumatorio de todos ellos.

11. Escriba un programa que vaya leyendo del teclado números enteros de forma iterativa hasta que el usuario introduzca uno positivo.

a) Finalmente deberá imprimir un mensaje mostrando el número positivo introducido por teclado. Por ejemplo, si el usuario introduce: "-12, -4, -8 y 10", el programa debe mostrar el siguiente mensaje: "El número positivo introducido ha sido 10".

b) Complete el código para que el programa cuente cuántos números negativos ha introducido el usuario hasta que tecleó un número positivo. Finalmente imprimirá el mensaje con el valor de

dicho contador. Por ejemplo, si el usuario introduce: "-10 -5 -7 9", entonces el programa debe mostrar el siguiente mensaje: "El usuario introdujo 3 números negativos antes de introducir el primer número positivo cuyo valor ha sido 9".

12. Escriba un programa que recoja un valor entero desde teclado y que muestre la tabla de multiplicar de dicho número.

13. Escriba un programa que recoja diez valores desde teclado y muestre posteriormente:

a) La media final.

b) El máximo de todos ellos (introduzca un condicional dentro del bucle).

c) El mínimo de todos ellos (introduzca un condicional dentro del bucle).

d) El porcentaje de positivos que se han introducido (contabilice el número de positivos y, a posteriori, calcule el porcentaje atendiendo al número total).

e) El porcentaje de negativos que se han introducido (contabilice el número de negativos y, a posteriori, calcule el porcentaje atendiendo al número total).

14. Escriba un programa que pida dos años, y muestre todos los años bisiestos comprendidos entre ambos. Un año es bisiesto si es múltiplo de 4 y no es múltiplo de 100, o bien es múltiplo de 400.

15. Analice y describa qué hace el siguiente fragmento de código:

```
int main(void){
	int n = 0;
	int m = 0;
	printf("Introduzca un número: ");
	scanf("%d", &n);

	while(n > 9){
		m = n % 10;
		n = n / 10;
		printf("%d\n", m);
	}
	printf("%d.", n);
	return 0;
}
```

Realice una traza del programa para los siguientes casos:

- Siendo n un número positivo de 1 cifra.
- Siendo n un número positivo de 2 cifras.
- Siendo n un número positivo de 3 o más cifras.
- Siendo n un número negativo de 1 cifra.
- Siendo n un número negativo de 2 o más cifras.

¿Realizaría alguna modificación para que el programa funcionara para cualquier valor de n positivo o negativo?

16. Modifique el programa "calculadora" de la práctica anterior. La estructura formada por diversos `if-else` se mantendrá igual; pero, con motivo de poder permitir diversas operaciones sin necesidad

de relanzar el programa, se añadirá una acción adicional al menú ("0: SALIR"), de forma que mientras la condición indicada por el usuario no sea 0, se comprobará qué operación se realiza; y, en caso de serlo, se terminará la ejecución (haga uso de un `while` que englobe el conjunto de `if-else` que poseía).

17. Escriba un programa que calcule y muestre por pantalla el máximo común divisor (MCD) de dos números enteros a, b cuyos valores se leerán desde el teclado. El MCD de a, b es, como mínimo, 1 y como máximo, el menor de los dos valores a, b.

18. Escribe un programa que calcule y muestre por pantalla el mínimo común múltiplo (mcm) de dos números enteros a, b cuyos valores se leerán desde el teclado. El mcm de a, b es, como mínimo, el mayor de los dos valores a, b y como máximo, el producto de a*b.

19. Realice un programa que indique si un número es primo (imprima por pantalla "S") o no (imprima por pantalla "N"), comprobando si es divisible por los números naturales inferiores a él (excepto 0, 1 y él mismo).

20. Escriba un programa que calcule el resto de la división entera entre dos números a y b, enteros positivos dados por el usuario, sin usar los operadores / ni %.

21. Escriba un programa que imprima los 15 primeros términos de la sucesión de *Fibonacci* por pantalla. Los dos primeros términos de dicha sucesión son 1 y 2 y los siguientes términos se van calculando a partir de la suma de los dos anteriores, es decir, el término *n* de la sucesión (para *n*>=3) viene dado por la suma de los dos términos inmediatamente anteriores a él:

$$x_1 = 1 \quad x_2 = 1$$
$$x_n = x_{n-1} + x_{n-2} \quad para\ n >= 3$$

22. Imprima por pantalla la descomposición factorial de un número. (Nota: compruebe primero si el número es divisible por 2; si lo es, imprimir un 2 y seguir iterando para buscar factores pero con el numero=numero/2. Cuando se hayan sacado todas las veces que es divisible por 2, repetir todo el proceso para los números impares).