

## LABORATORIO 3

### ESTRUCTURAS REPETITIVAS WHILE / DO WHILE

#### 1 OBJETIVOS

Al finalizar esta actividad, serás capaz de:

- Entender el funcionamiento de las estructuras iterativas en general; la(s) **condición(es) para finalizar**, que ha de ejecutarse en cada vuelta, **la(s) actualización(es)** de las variables para que en alguna de las iteraciones se produzca la salida de la estructura iterativa, etc.
- Entender la diferencia existente entre utilizar un while o un do-while.

#### 2 MOTIVACIÓN

La iteración es una estructura básica de casi todos los lenguajes de programación. Nos permite resolver programas complejos a través de la repetición de acciones más sencillas.

El while en C

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

**Mientras** la condición sea cierta **ejecutar una y otra vez** las instrucciones que aparecen entre llaves

El Do-while en C.

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

**Ejecutar una vez** la instrucciones que aparecen entre llaves y luego **Mientras** la condición sea cierta **ejecutarlas una y otra vez**.

#### 3 EJERCICIO EJEMPLO

(lo encontraremos en los apuntes también)

Programa que presenta en pantalla los primeros n números naturales. El programa deberá pedir al usuario que introduzca un número n tal que ( $n \geq 1$  y  $n \leq 10$ ). El programa deberá imprimir los número desde 1 hasta n. Si el usuario introdujese un valor para n que no cumpliese 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>
#include <stdlib.h>


void main()
{
    int n, num;

    printf("\nIntroduce un número entre 1 y 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 acciones a repetir*/

        } /*el final del else*/
    system("PAUSE");
}

```



## 4 EJERCICIOS

### 4.1 Ejercicio 1

#### 4.1.1 Objetivo:

El objetivo del ejercicio 1 consiste en experimentar con soluciones de "fuerza bruta". "fuerza bruta" en el sentido de que consiste en generar todas las posibilidades (es decir consiste en presuponer que todo número  $< n$  es en principio un *posible divisor*). filtrando (o imprimiendo en este caso) las que nos resultan interesantes (es decir los que sean divisores).

#### 4.1.2 Ayuda para el enunciado 1:

Para descubrir cuales son los divisores de un número  $n$  generaremos de forma ciega **todos** los números desde el 1 a  $n$  (o  $n/2$ ) y por cada número generado comprobaremos si es divisor del número que nos dan.

Utilizar el ejemplo inicial para ir generando los números. La diferencia consistirá en que por cada número que se genere (en cada vuelta del while se generará 1), tendremos que decidir si es divisor o no. Para saber si un número es divisor de otro utilizar la operación aritmética %, que nos calcula el resto de una división, si  $a \% b = 0$ , entonces podremos decir que  $b$  es divisor de  $a$ ).

#### 4.1.3 Enunciado 1

Pedir un número al usuario y determinar cuales son sus divisores.

Ejemplo de pantalla .

14: sus divisores son el 7, el 2 y el 1

## 4.2 Ejercicio 2

### 4.2.1 Objetivo:

El objetivo del ejercicio 2 consiste en además de experimentar con soluciones de "fuerza bruta", introducir el concepto de *flag* (*bandera o chivato*). "fuerza bruta" otra vez, porque generaremos todos los posibles divisores y en este caso *marcando* (a través de uso de un *flag o chivato*) las que nos resultan interesantes (en este caso los que sean divisores entre 2 y n-1). Existen varias versiones para solucionar este ejercicio, algunas pueden ser más *eficientes* que otras.

### 4.2.2 Ayuda para el enunciado 2:

Para descubrir si un número es primo generaremos de forma ciega **todos** los números desde el 2 a n-1 y por cada número generado comprobaremos si es divisor del número que nos dan. Si en ese rango [2,n-1] algún número fuese divisor, entonces el número no sería primo.

Utilizar el ejemplo anterior para ir generando los posibles divisores, pero en este caso partiendo del 2 como primer posible divisor.

### 4.2.3 Enunciado 2

Hacer un programa que permita que el usuario que introduzca un número que determine si es primo o no.

Ejemplo de lo que debería aparecer en pantalla si el usuario introdujese un 13 o bien un 14.

13, el 13 es primo  
14, el 14 no es primo

(Recordemos que un número es primo cuando sólo es divisible por sí mismo y por la unidad).

## 4.3 Ejercicio 3

### 4.3.1 Objetivo:

El objetivo del ejercicio 2 consiste ver cual es la opción más apropiada, si utilizar **While** o **Do-while**.

### 4.3.2 Ayuda para el enunciado 3:

Recordar que para leer un carácter haremos

```
fflush(stdin);  
scanf("%c",&variable);
```

y recordar también que si queremos comprobar si el usuario ha introducido una s deberemos escribirla entre comillas simples 's'.

La condición debería de parecerse a esto

(variable == 's')

Utilizar el ejemplo anterior para descubrir cuando un número es primo.

### 4.3.3 Enunciado 3

Hacer un programa que permita que el usuario introduzca un número y el programa determine si el número es primo o no, y después le pregunte si quiere seguir introduciendo más números ("¿Quieres seguir introduciendo números?(s/n)") de forma que se le vayan pidiendo números mientras el usuario responda afirmativamente (s).

Ejemplo de lo que debería aparecer en pantalla si el usuario introdujese un 13 o bien un 14.

```
Introduce un número:13
el 13 es primo
¿Quieres seguir introduciendo más números?(s/n): s
Introduce un número:14
el 14 no es primo
¿Quieres seguir introduciendo más números?(s/n): n
```

(Recordemos que un número es primo cuando sólo es divisible por sí mismo y por la unidad).

## 4.4 Ejercicio 4

### 4.4.1 Objetivo:

El objetivo del ejercicio 4 es aprender a desglosar un número en sus distintas cifras, para lo cual será muy importante fijarnos que el hacer la operación  $\text{num}/10$  no modifica la variable num, (por ejemplo  $a=\text{num}/10$ , aquí num no queda modificado) a no ser que explícitamente así lo decidamos (comparar  $a=\text{num}/10$  con  $\text{num}=\text{num}/10$ ).

Lo dicho, la estrategia en este caso será desglosar un número en sus distintas cifras ( a través de la iteración).

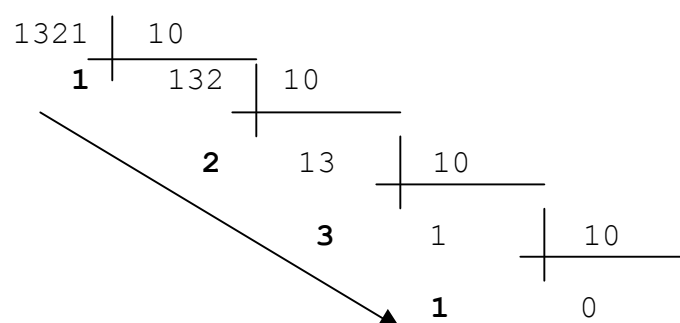
Pasos: Ir consiguiendo las cifras que forman el número para poder al mismo tiempo construyendo el mismo número dado la vuelta. Así finalmente sabremos si el número inicial es capicúa comparando si el número inicial y el mismo número dado la vuelta SON IGUALES.

1221 dado la vuelta es 1221 como son iguales es capicúa

1321 dado la vuelta es 1231 como no son iguales no son capicúa

### 4.4.2 Ayuda para el enunciado 4:

Para dar la vuelta al número



el número dado la vuelta se forma cogiendo los restos y acumulándolos y es 1231

#### 4.4.3 Enunciado 4

Pedir un número al usuario y determinar si es capicúa.

Ejemplo de lo que debería aparecer por pantalla si el usuario introdujese 1221 o bien 1 o 1321

```
1221 es capicúa
1 es capicúa
1321 no es capicúa
```

#### 4.5 Ejercicio 5

##### 4.5.1 Objetivo:

Ver que calcular el máximo de una secuencia se puede ir haciendo a medida que el usuario va introduciendo los números sin necesidad de almacenarlos todos aplazándolo hasta que todos los números hayan sido introducidos.

##### 4.5.2 Ayuda para el enunciado 5:

utilizaremos una variable que ira manteniendo el máximo que se vaya encontrando HASTA EL MOMENTO. Una variable que bien podríamos llamar max\_hasta\_el\_momento.

Por ejemplo:

```
Numero introducido 5 max_hasta_el_momento 5
Numero introducido 7 max_hasta_el_momento 7
Numero introducido 10 max_hasta_el_momento 10
Numero introducido 2 max_hasta_el_momento 10
Numero introducido -4 max_hasta_el_momento 10
Numero introducido -15 max_hasta_el_momento 10
Numero introducido 15 max_hasta_el_momento 15
Numero introducido 8 max_hasta_el_momento 15
Numero introducido 9 max_hasta_el_momento 15
Numero introducido 0 max_hasta_el_momento 15
```

##### 4.5.3 Enunciado 5

Hacer un programa que permita que el usuario vaya introduciendo números (cuando el usuario introduzca un 0 el programa acabará). Nuestro programa deberá sacar por pantalla el número máximo de entre los números introducidos por el usuario.

Ejemplo. Si el usuario a metido -1 3 4 -2 6 1000 2000, nuestro programa debería imprimir el 2000.