CAPÍTULO 3

FUNCIONES DE ENTRADA Y SALIDA POR CONSOLA

Hasta el momento, hemos presentado las sentencias de creación y declaración de variables. También hemos visto multitud de operaciones que se pueden realizar con las diferentes variables y literales. Pero aún no sabemos cómo mostrar un resultado por pantalla. Y tampoco hemos aprendido todavía a introducir información, para un programa en ejecución, desde el teclado.

El objetivo de este breve capítulo es iniciar en la comunicación entre el programa y el usuario.

Lograr que el valor de una variable almacenada de un programa sea mostrado por pantalla sería una tarea compleja si no fuese porque ya ANSI C ofrece funciones que realizan esta tarea. Y lo mismo ocurre cuando el programador quiere que sea el usuario quien teclee una entrada durante la ejecución del programa.

Estas funciones, de entrada y salida estándar de datos por consola, están declaradas en un archivo de cabecera llamado **stdio.h**. Siempre que deseemos usar estas funciones deberemos añadir, al principio del código de nuestro programa, la directriz ***#include <stdio.h>***.

Salida de datos. La función *printf()*.

El prototipo de la función es el siguiente:

int printf(const char *cadena_control[, argumento, ...]);

Qué es un **prototipo** de una función es cuestión que habrá que explicar en otro capítulo. Sucintamente, diremos que el prototipo indica el modo de empleo de la función: qué tipo de dato devuelve y qué valores espera recibir cuando se hace uso de ella. El prototipo nos sirve para ver cómo se emplea esta función.

La función *printf* devuelve un valor entero. Se dice que es de tipo ***int***. La función *printf* devuelve un entero que indica el número de bytes que ha impreso en pantalla. Si, por la causa que sea, la función no se ejecuta correctamente, en lugar de ese valor entero lo que devuelve es un valor que significa error (por ejemplo un valor negativo). No descendemos a más detalles.

La función, como toda función, lleva después del nombre un par de paréntesis. Entre ellos va redactado el texto que deseamos que quede impreso en la pantalla. La *cadena_control* indica el texto que debe ser impreso, con unas especificaciones que indican el formato de esa impresión; es una cadena de caracteres recogidos entre comillas, que indica el texto que se ha de mostrar por pantalla. A lo largo de este capítulo aprenderemos a crear esas cadenas de control que especifican la salida y el formato que ha de mostrar la función *printf*.

Para comenzar a practicar, comenzamos por escribir en el editor de C el siguiente código. Es muy recomendable que a la hora de estudiar

cualquier lenguaje de programación, y ahora en concreto el lenguaje C, se trabaje delante de un ordenador que tenga un editor y un compilador de código:

```
#include <stdio.h>
void main(void)
{
      printf(“Texto a mostrar en pantalla”);
}
```

Que ofrecerá la siguiente salida por pantalla

```
Texto a mostrar en pantalla
```

Y así, cualquier texto que se escriba entre las comillas aparecerá en pantalla.

Si introducimos ahora otra instrucción con la función *printf* a continuación y debajo de la otra, por ejemplo

```
printf(“Otro texto”);
```

Entonces lo que tendremos en pantalla será

```
Texto a mostrar en pantallaOtro texto
```

Y es que la función *printf* continua escribiendo donde se quedó la vez anterior.

Muchas veces nos va a interesar introducir, en nuestra cadena de caracteres que queremos imprimir por pantalla, algún carácter de, por ejemplo, salto de línea. Pero si tecleamos la tecla intro en el editor de C lo que hace el cursor en el editor es cambiar de línea y eso que no queda reflejado luego en el texto que muestra el programa en tiempo de ejecución.

Para poder escribir este carácter de salto de línea, y otros que llamamos caracteres de control, se escribe, en el lugar de la cadena donde queremos que se imprima ese carácter especial, una barra invertida (‘\’) seguida de una letra. Cuál letra es la que se debe poner dependerá de qué carácter especial se desea introducir. Esos caracteres de control son

caracteres no imprimibles, o caracteres que tienen ya un significado especial en la función *printf*.

Por ejemplo, el código anterior quedaría mejor de la siguiente forma:

```
#include <stdio.h>
void main(void)
{
      printf("Texto a mostrar en pantalla\n");
      printf("Otro texto.")
}
```

que ofrecerá la siguiente salida por consola:

```
Texto a mostrar en pantalla
Otro texto
```

Ya que al final de la cadena del primer *printf* hemos introducido un carácter de control salto de línea: "\n" significa, dentro de la función *printf*, salto de línea.

Las demás letras con significado para un carácter de control en esta función vienen recogidas en la tabla 3.1.

\a	Carácter sonido. Emite un pitido breve.
\v	Tabulador vertical.
\0	Carácter nulo.
\n	Nueva línea.
\t	Tabulador horizontal.
\b	Retroceder un carácter.
\r	Retorno de carro.
\f	Salto de página.
\'	Imprime la comilla simple.
\"	Imprime la comilla doble.
****	Imprime la barra invertida '\'.
\xdd	dd es el código ASCII, en hexadecimal, del carácter que se desea imprimir.

Tabla 3.1.: Caracteres de control en la cadena de la función *printf*.

Muchas pruebas se pueden hacer ya en el editor de C, para compilar y ver la ejecución que resulta. Gracias a la última opción de la tabla 3.1. es posible imprimir todos los caracteres ASCII y los tres inmediatamente

anteriores sirven para imprimir caracteres que tienen un significado preciso dentro de la cadena de la función *printf*. Gracias a ellos podemos imprimir, por ejemplo, un carácter de comillas dobles evitando que la función *printf* interprete ese carácter como final de la cadena que se debe imprimir.

El siguiente paso, una vez visto cómo imprimir texto prefijado, es imprimir en consola el valor de una variable de nuestro programa.

Cuando en un texto a imprimir se desea intercalar el valor de una variable, en la posición donde debería ir ese valor se coloca el carácter '%' seguido de algunos caracteres. Según los caracteres que se introduzcan, se imprimirá un valor de un tipo de dato concreto, con un formato de presentación determinado. Ese carácter '%' y esos caracteres que le sigan son los **especificadores de formato**. Al final de la cadena, después de las comillas de cierre, se coloca una coma y el nombre de la variable que se desea imprimir.

Por ejemplo, el siguiente código

```
#include <stdio.h>
void main(void)
{
      short int a = 5 , b = 10 , c;
      c = a + b++;
      printf("Ahora c vale %hd \n",c);
      printf("y b vale ahora %hd ",b);
}
```

Que ofrece la siguiente salida por pantalla:

```
Ahora c vale 15
y b vale ahora 11
```

Una cadena de texto de la función *printf* puede tener tantos especificadores de formato como se desee. Tantos como valores de variables queramos imprimir por pantalla. Al final de la cadena, y después de una coma, se incluyen tantas variables, separadas también por comas, como especificadores de formato se hayan incluido en la cadena de texto. Cada grupo de caracteres encabezado por % en el

primer argumento de la función (la cadena de control) está asociado con el correspondiente segundo, tercero, etc. argumento de esa función. Debe existir una correspondencia biunívoca entre el número de especificadores de formato y el número de variables que se recogen después de la cadena de control; de lo contrario se obtendrán resultados imprevisibles y sin sentido.

El especificador de formato instruye a la función sobre la forma en que deben ir impresos cada uno de los valores de las variables que deseamos que se muestren por pantalla.

Los especificadores tienen la forma:

%[flags][ancho campo][.precisión][F/N/h/l/L] type

Veamos los diferentes componentes del especificador de formato:

type: Es el único argumento necesario. Consiste en una letra que indica el **tipo de dato** a que corresponde al valor que se desea imprimir en esa posición. En la tabla 3.2. se recogen todos los valores que definen tipos de dato. Esta tabla está accesible en las ayudas de editores y compiladores de C.

- ***[F / N / h / l / L]***: Estas cinco letras son **modificadores de tipo** y preceden a las letras que indican el tipo de dato que se debe mostrar por pantalla.

  La letra **h** es el modificador ***short*** para valores enteros.

  La letra **l** tiene dos significados: es el modificador ***long*** para valores enteros. Y, precediendo a la letra f indica que allí debe ir un valor de tipo ***double***.

  La letra **L** precediendo a la letra f indica que allí debe ir un valor de tipo ***long double***.

- ***[ancho campo][.precisión]***: Estos dos indicadores opcionales deben ir antes de los indicadores del tipo de dato. Con el **ancho de campo**, el

especificador de formato indica a la función *printf* la **longitud mínima** que debe ocupar la impresión del valor que allí se debe mostrar.

%d	Entero con signo, en base decimal.
%i	Entero con signo, en base decimal.
%o	Entero (con o sin signo) codificado en base octal.
%u	Entero sin signo, en base decimal.
%x	Entero (con o sin signo) codificado en base hexadecimal, usando letras minúsculas. Codificación interna de los enteros.
%X	Entero (con o sin signo) codificado en base hexadecimal, usando letras mayúsculas. Codificación interna de los enteros.
%f	Número real con signo.
%e	Número real con signo en formato científico, con el exponente 'e' en minúscula.
%E	Número real con signo en formato científico, con el exponente 'e' en mayúscula.
%g	Número real con signo, a elegir entre formato e ó f según cuál sea el tamaño más corto.
%G	Número real con signo, a elegir entre formato E ó f según cuál sea el tamaño más corto.
%c	Un carácter. El carácter cuyo ASCII corresponda con el valor a imprimir.
%s	Cadena de caracteres.
%p	Dirección de memoria.
%n	No lo explicamos aquí ahora.
%%	Si el carácter % no va seguido de nada, entonces se imprime el carácter sin más.

Tabla 3.2.: Especificadores de tipo de dato en la función *printf*.

Para mostrar información por pantalla la función *printf* emplea un tipo de letra de paso fijo. Esto quiere decir que cada carácter impreso ocasiona el mismo desplazamiento del cursor hacia la derecha. Al decir que el ancho de campo indica la longitud mínima se quiere decir que este parámetro señala cuántos avances de cursor deben realizarse, como mínimo, al imprimir el valor.

Por ejemplo, las instrucciones

```
long int a = 123, b = 4567, c = 135790;
printf("La variable a vale ... %6li.\n",a);
printf("La variable b vale ... %6li.\n",b);
printf("La variable c vale ... %6li.\n",c);
```

tiene la siguiente salida:

```
La variable a vale ...    123.
La variable b vale ...   4567.
La variable c vale ... 135790.
```

donde vemos que los tres valores impresos en líneas diferentes quedan alineados en sus unidades, decenas, centenas, etc. gracias a que todos esos valores se han impreso con un ancho de campo igual a 6: su impresión ha ocasionado tantos desplazamientos de cursos como indica el ancho de campo.

Si la cadena o número es mayor que el ancho de campo indicado ignorará el formato y se emplean tantos pasos de cursor como sean necesarios para imprimir correctamente el valor.

Si se desea, es posible **rellenar con ceros los huecos del avance** de cursor. Para ellos se coloca un 0 antes del número que indica el ancho de campo

La instrucción

```
printf("La variable a vale ... %06li.\n",a);
```

ofrece como salida la siguiente línea en pantalla:

```
La variable a vale ... 000123.
```

El **parámetro de precisión** se emplea para valores con coma flotante. Indica el número de decimales que se deben mostrar. Indica cuántos dígitos no enteros se deben imprimir: las posiciones decimales. A ese valor le precede un punto. Si el número de decimales del dato almacenado en la variable es menor que la precisión señalada, entonces la función *printf* completa con ceros ese valor. Si el número de decimales del dato es mayor que el que se indica en el parámetro de precisión, entonces la función *printf* trunca el número.

Por ejemplo, el código

```
double raiz_2 = sqrt(2);
printf("A. Raiz de dos vale %lf\n",raiz_2);
printf("B. Raiz de dos vale %12.1lf\n",raiz_2);
printf("C. Raiz de dos vale %12.3lf\n",raiz_2);
printf("D. Raiz de dos vale %12.5lf\n",raiz_2);
printf("E. Raiz de dos vale %12.7lf\n",raiz_2);
printf("F. Raiz de dos vale %12.9lf\n",raiz_2);
printf("G. Raiz de dos vale %12.11lf\n",raiz_2);
printf("H. Raiz de dos vale %5.7lf\n",raiz_2);
printf("I. Raiz de dos vale %012.4lf\n",raiz_2);
```

que ofrece la siguiente salida por pantalla:

```
A. Raiz de dos vale 1.414214
B. Raiz de dos vale          1.4
C. Raiz de dos vale        1.414
D. Raiz de dos vale      1.41421
E. Raiz de dos vale    1.4142136
F. Raiz de dos vale  1.414213562
G. Raiz de dos vale 1.41421356237
H. Raiz de dos vale 1.4142136
I. Raiz de dos vale 0000001.4142
```

La función ***sqrt*** está declarada en el archivo de cabecera **math.h**. Esta función devuelve la raíz cuadrada (en formato ***double***) del valor (también ***double***) que ha recibido como parámetro de entrada, entre paréntesis.

Por defecto, se toman seis decimales, sin formato alguno. Se ve en el ejemplo el truncamiento de decimales. En el caso G., la función *printf* hace caso omiso del ancho de campo pues se exige que muestre un valor que tiene un carácter para la parte entera, otro para el punto decimal y once para los decimales: en total 13 caracteres, y no 12 como señala en ancho de campo. y es que el punto decimal es un carácter más dentro de la impresión por pantalla del valor.

- ***[flags]***: Son caracteres que introducen unas últimas modificaciones en el modo en que se presenta el valor. Algunos de sus valores y significados son:

  carácter ‘–‘: el valor queda justificado hacia la izquierda.

carácter +: el valor se escribe con signo, sea éste positivo o negativo. En ausencia de esta bandera, la función *printf* imprime el signo únicamente si es negativo.

carácter en blanco: Si el valor numérico es positivo, deja un espacio en blanco. Si es negativo imprime el signo.

Existen otras muchas funciones que muestran información por pantalla. Muchas de ellas están definidas en el archivo de cabecera **stdio.h**. Con la ayuda a mano, es sencillo aprender a utilizar muchas de ellas.

Entrada de datos. La función *scanf()*.

La función *scanf* de nuevo la encontramos declarada en el archivo de cabecera **stdio.h**. Permite la entrada de datos desde el teclado. La ejecución del programa queda suspendida en espera de que el usuario introduzca un valor y pulse la tecla de validación (intro).

La ayuda de cualquier editor y compilador de C es suficiente para lograr hacer un buen uso de ella. Presentamos aquí unas nociones básicas, suficientes para su uso más habitual. Para la entrada de datos, al igual que ocurría con la salida, hay otras funciones válidas que también pueden conocerse a través de las ayudas de los distintos editores y compiladores de C.

El prototipo de la función es:

int scanf(const char *cadena_control[,direcciones,…]);

La función *scanf* puede leer del teclado tantas entradas como se le indiquen. De todas formas, se recomienda usar una función *scanf* para cada entrada distinta que se requiera.

El valor que devuelve la función es el del número de entradas diferentes que ha recibido. Si la función ha sufrido algún error, entonces devuelve un valor que significa error (por ejemplo, un valor negativo).

En la cadena de control se indica el tipo de dato del valor que se espera recibir por teclado. No hay que escribir texto alguno en la cadena de control de la función *scanf*: únicamente el especificador de formato.

El formato de este especificador es similar al presentado en la función *printf*: un carácter % seguido de una o dos letras que indican el tipo de dato que se espera. Luego, a continuación de la cadena de control, y después de una coma, se debe indicar **dónde** se debe almacenar ese valor: la posición de una variable que debe ser del mismo tipo que el indicado en el especificador. El comportamiento de la función *scanf* es imprevisible cuando no coinciden el tipo señalado en el especificador y el tipo de la variable; en ese caso, habitualmente aborta la ejecución del programa.

Las letras que indican el tipo de dato a recibir se recogen en la tabla 3.3. Los modificadores de tipo de dato son los mismos que para la función *printf*.

%d	Entero con signo, en base decimal.
%i	Entero con signo, en base decimal.
%o	Entero codificado en base octal.
%u	Entero sin signo, en base decimal.
%x	Entero codificado en base hexadecimal, usando letras minúsculas. Codificación interna de los enteros.
%X	Entero codificado en base hexadecimal, usando letras mayúsculas. Codificación interna de los enteros.
%f	Número real con signo.
%e	Número real con signo en formato científico, con el exponente ‘e’ en minúscula.
%c	Un carácter. El carácter cuyo ASCII corresponda con el valor a imprimir.
%s	Cadena de caracteres.
%p	Dirección de memoria.
%n	No lo explicamos aquí ahora.

Tabla 3.3.: Especificadores de tipo de dato en la función *scanf*.

La cadena de control tiene otras especificaciones, pero no las vamos a ver aquí. Se pueden obtener el la ayuda del compilador.

Además de la cadena de control, la función *scanf* requiere de otro parámetro: el lugar dónde se debe almacenar el valor introducido. **La función *scanf* espera, como segundo parámetro, el lugar donde se aloja la variable, no el nombre**. Espera la dirección de la variable. Así está indicado en su prototipo.

Para poder saber la dirección de una variable, C dispone de un operador unario: &. El operador dirección, prefijo a una variable, devuelve la dirección de memoria de esta variable. El olvido de este operador en la función *scanf* es frecuente en programadores noveles. Y de consecuencias desastrosas: siempre ocurre que el dato introducido no se almacena en la variable que deseábamos, alguna vez producirá alteraciones de otros valores y las más de las veces llevará a la inestabilidad del sistema y se deberá finalizar la ejecución del programa.

Recapitulación.

Hemos presentado el uso de las funciones *printf()* y *scanf()*, ambas declaradas en el archivo de cabecera ***stdio.h***. Cuando queramos hacer uno de una de las dos funciones, o de ambas, deberemos indicarle al programa con la directiva de preprocesador ***#include <stdio.h>***.

El uso de ambas funciones se aprende en su uso habitual. Los ejercicios del capítulo anterior pueden ayudar, ahora que ya las hemos presentado, a practicar con ellas.

Ejercicios.

16. ***Escribir un programa que muestre al código ASCII de un carácter introducido por teclado.***

```
#include <stdio.h>

void main(void)
{
      unsigned char ch;

      printf("Introduzca un carácter por teclado ... ");
      scanf("%c",&ch);

      printf("El carácter introducido ha sido %c\n",ch);
      printf("Su código ASCII es el %hd", ch);
}
```

Primero mostramos el carácter introducido con el especificador de tipo %c: así muestra el carácter por pantalla. Y luego mostramos el mismo valor de la variable *ch* con el especificador %hd, es decir, como entero corto, y entonces nos muestra el valor numérico de ese carácter.

17. ***Lea el programa siguiente, e intente explicar la salida que ofrece por pantalla.***

```
#include <stdio.h>

void main(void)
{
      signed long int sli;
      signed short int ssi;

      printf("Introduzca un valor negativo para sli ... ");
      scanf("%ld",&sli);

      printf("Introduzca un valor negativo para ssi ... ");
      scanf("%hd",&ssi);
```

```
      printf("El valor sli es %ld\n",sli);
      printf("El valor ssi es %ld\n\n",ssi);

      printf("El valor sli como \"%%lX\" es %lX\n",sli);
      printf("El valor ssi como \"%%hX\" es %hX\n\n",ssi);

      printf("El valor sli como \"%%lu\" es %lu\n",sli);
      printf("El valor ssi como \"%%hu\" es %hu\n\n",ssi);

      printf("El valor sli como \"%%hu\" es %hu\n",sli);
      printf("El valor ssi como \"%%lu\" es %lu\n\n",ssi);

      printf("El valor sli como \"%%hi\" es %hi\n",sli);
      printf("El valor ssi como \"%%li\" es %li\n\n",ssi);
}
```

La salida que ha obtenido su ejecución es la siguiente:

```
Introduzca un valor negativo para sli ... -23564715
Introduzca un valor negativo para ssi ... -8942

El valor sli es -23564715
El valor ssi es -8942

El valor sli como "%lX" es FE986E55
El valor ssi como "%hX" es DD12

El valor sli como "%lu" es 4271402581
El valor ssi como "%hu" es 56594

El valor sli como "%hu" es 28245
El valor ssi como "%lu" es 4294958354

El valor sli como "%hi" es 28245
El valor ssi como "%li" es -8942
```

Las dos primeras líneas no requieren explicación alguna: recogen las entradas que se han introducido por teclado cuando se han ejecutado las instrucciones de la función *scanf()*. Las dos siguientes tampoco: muestran por pantalla lo valores introducidos: el primero (*sli*) es ***long int***, y se muestra con el especificador de formato %ld ó %li; el segundo (*ssi*) es ***short int***, y se muestra con el especificador de formato %hd ó %hi.

Las siguientes líneas de salida son:

```
El valor sli como "%lX" es FE986E55
```

```
El valor ssi como "%hX" es DD12
```

Que muestran los números tal y como los tiene codificados el ordenador: al ser enteros con signo, y ser negativos, codifica el bit más significativo (el bit 31 en el caso de *sli*, el bit 15 en el caso de *ssi*) a uno porque es el bit del signo; y codifica el resto de los bits (desde el bit 30 al bit 0 en el caso de *sli*, desde el bit 14 hasta el bit 0 en el caso de *ssi*) como el complemento a la base del valor absoluto del número codificado.

El número $(8942)_{10} = (22EE)_{16}$ se codifica, como número negativo (dígito 15 a 1 y resto el valor en su complemento a la base), de la forma DD12 . Y el número $(23564715)_{10} = (167\ 91AB)_{16}$ se codifica, como número negativo, de la forma FE98 6E55 .

Las dos siguientes líneas son:

```
El valor sli como "%lu" es 4271402581
El valor ssi como "%hu" es 56594
```

Muestra el contenido de la variable *lsi* que considera ahora como entero largo sin signo. Y por tanto toma esos 32 bits, que ya no los considera como un bit de signo y 31 de complemento a la base del número negativo, sino 32 bits de valor positivo codificado en binario: $(FE98\ 6E55)_{16} = (4.271.402.581)_{10}$.

Y muestra el contenido de la variable *ssi* que considera ahora como entero corto sin signo. Y por tanto toma esos 16 bits, que ya no los considera como un bit de signo y 15 de complemento a la base del número negativo, sino 16 bits de valor positivo codificado en binario: $(DD12)_{16} = (56.594)_{10}$.

Las dos siguientes líneas son:

```
El valor sli como "%hu" es 28245
El valor ssi como "%lu" es 4294958354
```

La primera de ellas considera la variable *sli* como una variable corta de 16 bits. Por tanto lo que hace es tomar los 16 bits menos significativos

de la variable de 32 bits y los interpreta como entero corto sin signo: $(6E55)_{16} = (28.245)_{10}$.

La segunda línea es de difícil interpretación: en realidad muestra un valor numérico que, expresado en base hexadecimal, es igual a $(FFFF\ DD12)_{16}$: ha añadido, delante de la variable de 16 bits, otros 16 bits que ha encontrado, casualmente, codificados en todos los bits a uno.

El último bloque es fácilmente interpretable una vez explicadas las dos líneas anteriores. Se deja al lector esa interpretación.

18. ***Escriba el siguiente programa y compruebe cómo es la salida que ofrece por pantalla.***

```
#include <stdio.h>
#include <math.h>
void main(void)
{
      double a = M_PI;

      printf(" 1. El valor de Pi es ... %20.1lf\n",a);
      printf(" 2. El valor de Pi es ... %20.2lf\n",a);
      printf(" 3. El valor de Pi es ... %20.3lf\n",a);
      printf(" 4. El valor de Pi es ... %20.4lf\n",a);
      printf(" 5. El valor de Pi es ... %20.5lf\n",a);
      printf(" 6. El valor de Pi es ... %20.6lf\n",a);
      printf(" 7. El valor de Pi es ... %20.7lf\n",a);
      printf(" 8. El valor de Pi es ... %20.8lf\n",a);
      printf(" 9. El valor de Pi es ... %20.9lf\n",a);
      printf("10. El valor de Pi es ... %20.10lf\n",a);
      printf("11. El valor de Pi es ... %20.11lf\n",a);
      printf("12. El valor de Pi es ... %20.12lf\n",a);
      printf("13. El valor de Pi es ... %20.13lf\n",a);
      printf("14. El valor de Pi es ... %20.14lf\n",a);
      printf("15. El valor de Pi es ... %20.15lf\n",a);
}
```

La salida que ofrece por pantalla es la siguiente:

```
 1. El valor de Pi es ... ·················3.1
 2. El valor de Pi es ... ·················3.14
```

```
 3. El valor de Pi es ... ··············3.142
 4. El valor de Pi es ... ··············3.1416
 5. El valor de Pi es ... ··············3.14159
 6. El valor de Pi es ... ··············3.141593
 7. El valor de Pi es ... ··············3.1415927
 8. El valor de Pi es ... ··········3.14159265
 9. El valor de Pi es ... ·········3.141592654
10. El valor de Pi es ... ·····3.1415926536
11. El valor de Pi es ... ·····3.14159265359
12. El valor de Pi es ... ·····3.141592653590
13. El valor de Pi es ... ·····3.1415926535898
14. El valor de Pi es ... ····3.14159265358979
15. El valor de Pi es ... ···3.141592653589793
```

Donde hemos cambiado los espacios en blanco por puntos en la parte de la impresión de los números. Y donde el archivo de biblioteca ***math.h*** contiene el valor del número pi, en la constante o identificador ***M_PI***. Efectivamente, emplea 20 espacios de carácter de pantalla para mostrar cada uno de los números. Y cambia la posición de la coma decimal, pues cada línea exigimos a la función *printf()* que muestre un decimal más que en la línea anterior.

19. ***Escriba un programa que genere 20 números de coma flotante de forma aleatoria en un rango de valores entre 0 y 1 millón, y los muestre, uno debajo de otro, alineados por la coma decimal.***

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
      randomize();

   for(int i = 1 ; i <= 20 ; i++)
      printf("%2d. %10.5lf\n",i,
            (double)random(1000000) / random (10000));
}
```

La función *random(n)* genera un entero aleatorio entre 0 y n – 1. La función *randomize()* debe invocarse siempre en aquellos programas que luego se vaya a usar el generador de aleatorios y sirve para inicializar el generador *random()*. Ambas funciones vienen declaradas en el archivo de cabecera ***stdlib.h***.

Aún no hemos llegado al capítulo de las estructuras de control, pero podemos entender por ahora, viendo el código, que este programa realice veinte veces la operación de generar dos enteros, dividirlos en cociente de coma flotante (por eso, previo al cociente, convertimos el dividendo en ***double***) y mostramos los distintos resultados, uno debajo del otro, con el especificador de tipo de dato *%10.5lf*. Así, si los números generados son menores que 100.000 quedarán en línea, como se ve en la salida que se muestra a continuación:

```
 1.   57.21568
 2.   62.41973
 3.  147.16501
 4.  120.04998
 5.  215.02813
 6.   52.82802
 7.  260.75406
 8.  721.83456
 9.    9.85598
10.  150.42073
11.    7.11266
12.   78.85494
13.   73.41685
14.  196.21048
15.   88.43795
16.  192.40674
17.  315.92087
18.   50.11689
19.    7.16849
20.  187.26519
```

Donde la coma decimal queda ubicada, en todos los números, en la misma columna. Y así también las unidades, decenas,... y decimales.

ANEXO: Ficha resumen de la función printf

printf <stdio.h>

int printf (*formato [, argumento , ...] **);***

Imprime los datos formateados a la salida estándar según especifica el argumento *formato*.

formato

Es una cadena de texto que contiene el texto que se va a imprimir. Opcionalmente puede contener etiquetas que se caracterizan por ir precedidas del carácter %. Estas etiquetas serán luego sustituidas cuando se ejecute la función por un valor especificado en los *argumentos*.

El número de etiquetas de formato debe ser el mismo que el de los argumentos que se indiquen. Cada etiqueta indica la ubicación donde se debe insertar un valor en la cadena de texto, y el formato en que se debe imprimir ese valor. Por cada etiqueta debe haber un argumento.

El formato de etiquetas sigue el siguiente prototipo:

%[flags][ancho][.precisión][modificadores]tipo

donde *tipo* es imprescindible.

tipo	***Salida***	**Ejemplo**
c	Carácter	**a**
d o **i**	Decimal entero con signo	**392**
u	Decimal entero sin signo	**7235**
o	Octal con signo	**610**
x	Entero hexadecimal sin signo	**7fa**
X	Entero hexadecimal sin signo (letras mayúsculas)	**7FA**
f	Decimal en punto flotante	**392.65**
e	Notación científica (mantisa/exponente) con el carácter **e**	**3.9265e2**
E	Notación científica (mantisa/exponente) con el carácter **E**	**3.9265E2**
g	Usa el que sea más corto entre %e y %f	**392.65**
G	Usa el que sea más corto entre %E y %f	**392.65**
s	Cadena de caracteres	**sample**
p	Dirección apuntada por el argumento	**B800:0000**

El resto de parámetros, esto es *flags*, *ancho, .precisión* y *modificadores* son opcionales y siguen el siguiente formato:

modificador	***significado***
h	*argumento* se interpreta como un **short int**.
l	*argumento* se interpreta como un **long int** cuando precede a un entero (d, i, o, u, x, X) o **double** si precede a un tipo de dato de coma flotante (f, g, G, e, E).
L	*argumento* se interpreta como un **long double** si precede a un tipo de dato de coma flotante (f, g, G, e, E).
ancho	***significado***
num	Especifica el número mínimo de caracteres que se imprimen. Si el valor que se imprime es menor que este *num* entonces el resultado es completado con ceros. El valor nunca es truncado incluso si es mayor.
*****	El ancho no se especifica en el *formato* de la cadena sino que se indica en el valor entero que precede al argumento que tiene que formatearse.
.precisión	***significado***
.*num*	Para los tipos **f, e, E, g, G:** indica el número de dígitos que se imprimen después del punto decimal (por defecto es **6**). Para el tipo **s**: indica el número máximo de caracteres que se imprimen. (por defecto imprime hasta encontrar el primer carácter null).
flags	***significado***
-	Alineación a la izquierda con el ancho de campo dado (por defecto alinea a la derecha). Este flag sólo tiene sentido cuando se especifica el ancho de campo.
+	Obliga a anteponer un signo al resultado (+ o -) si el tipo es con signo. (por defecto sólo el signo menos **-** se imprime).
blanco	Si el argumento es un valor positivo con signo, se inserta un blanco antes del número.
0	Coloca tantos ceros a la izquierda del número como sean necesarios para completar el ancho de campo especificado.
#	Usado con los tipos **o**, **x** o **X** el valor es precedido con **0**, **0x** o **0X** respectivamente si no son cero. Usado con **e**, **E** o **f** obliga a que el valor de salida contenga el punto decimal incluso aunque sólo sigan ceros. Usado con **g** o **G** el resultado es el mismo que con **e** o **E** pero los ceros sobrantes no se eliminan.

argumento(s)

Parámetro(s) opcional(es) que contiene(n) los datos que se insertarán en el lugar de los **%** etiquetas especificados en los parámetros del formato. Debe haber el mismo número de parámetros que de etiquetas de formato.

Valor de retorno de printf.

Si tiene éxito representa el número total de caracteres impresos. Si hay un error, se devuelve un número negativo.

Ejemplo.

```
/* printf: algunos ejemplos de formato*/
#include <stdio.h>

void main(void)
{
   printf("Caracteres: %c %c \n", 'a', 65);
   printf("Decimales: %d %ld\n", 1977, 650000);
   printf("Precedidos de blancos: %10d \n", 1977);
   printf("Precedidos de ceros: %010d \n", 1977);
   printf("Formato: %d %x %o %#x %#o\n",100,100,100,100,100);
   printf("float: %4.2f %+.0e %E\n", 3.1416, 3.1416, 3.1416);
   printf("Ancho: %*d \n", 5, 10);
   printf("%s \n", "Mi mamá me mima");
}
```

Y la salida:

```
Caracteres: a A
Decimales: 1977 650000
Precedidos con blancos:       1977
Precedidos con ceros: 0000001977
Formato: 100 64 144 0x64 0144
float: 3.14 +3e+000 3.141600E+000
Ancho:    10
Mi mamá me mima
```