Fundamentos de Programación – Programación Estructurada en C:
3.- FUNCIONES DE ENTRADA Y DE SALIDA

Maider Huarte Arrayago

Fundamentos de Programación – Programación Estructurada en C:

3.- FUNCIONES DE ENTRADA Y DE SALIDA

3.- FUNCIONES DE ENTRADA Y DE SALIDA

3.1.- INTRODUCCIÓN

Antes de seguir explicando la sintaxis de C, se van a mostrar algunas de las funciones de Entrada y de Salida más usadas, para que los ejemplos posteriores se comprendan de una forma más fácil, y ver cómo se puede comunicar el usuario con un programa.

Hay que tener en cuenta, que las funciones de Entrada y de Salida que se van a explicar son estándar. **Todas** están definidas en la librería correspondiente, que en el caso de máquinas MS DOS/Windows es la *stdio* (STandarD Input/Output) y en máquinas Linux, la *libc*. Así, para poder usarlas en los programas, será necesario **incluir** su fichero de cabecera, que en cualquiera de los casos es el ***stdio.h***, en la Sección de Descripción del programa (punto **2.1.- Comando *#include*** del tema **2.- EL PRECOMPILADOR**).

3.2.- FUNCIÓN *printf*

Esta función escribe en la salida estándar (que, el Sistema Operativo, por defecto, pasa su contenido a la pantalla), el texto y el valor de las expresiones que se le indiquen. Internamente, escribe lo indicado en el buffer de salida estándar, desde el que el Sistema Operativo lee y pasa a pantalla lo que haya en él, cuando lo considere oportuno.

Los buffers de entrada y de salida estándar son unas zonas de la Memoria Principal del computador (punto **1.2.- ESTRUCTURA INTERNA BÁSICA DE COMPUTADORES** del tema **1.- LENGUAJES DE PROGRAMACIÓN** de la Parte I de la asignatura), que el Sistema Operativo asocia a un dispositivo de entrada y de salida respectivamente. Estos dispositivos, son los que permiten al computador comunicarse con el usuario humano. El dispositivo de entrada estándar por defecto, es el teclado, y el de salida, la pantalla.

Prototipo:

```
int printf(<cadena de control>[,<valor 1>,[...]]);
```

cadena de control: Una Constante Cadena, texto entre comillas dobles.

Los puntos suspensivos indican que puede haber, después del primer argumento, cualquier número y tipo de valores, que se podrán indicar como constantes (estudiadas en el punto 1.3.1.2.- Constantes del tema **1.-**

INTRODUCCIÓN AL LENGUAJE C), variables (punto 1.3.1.1. Variables del tema **1.- INTRODUCCIÓN AL LENGUAJE C**) o incluso expresiones a calcular (que se estudiarán en el siguiente tema). Así, el único argumento obligatorio en *printf*, es *cadena de control*.

Esta función escribe en la salida estándar (internamente, en el buffer de salida) de forma literal el contenido del texto indicado en *cadena de control*, sustituyendo las subcadenas especiales que pudiera haber en ella, con los valores indicados después, siguiendo el orden en el que aparecen en la llamada. La sintaxis de las subcadenas especiales (entre otros) contiene los siguientes elementos:

%[**-**][<Anchura>][**.**<Precisión>]<Carácter de Conversión>

Todas las subcadenas especiales a sustituir, comienzan con el carácter '%' y terminan con un Carácter de Conversión. La siguiente tabla muestra los caracteres de conversión y su significado:

Carácter de Conversión	Tipo de argumento
d, i	Número entero, en notación decimal
c	Carácter
s	Cadena de caracteres
e	Número de coma flotante, en notación exponencial
f	Número de coma flotante, en notación decimal
g	La opción más corta entre e y f
u	Número entero sin signo, en notación decimal
o	Número entero sin signo, en notación octal
x	Número entero sin signo, en notación hexadecimal
Nota: Para argumentos de tipo *double*, antes del carácter de conversión se pone 'l', y para los de tipo short, 'h'.	

Lo importante es que debe haber correspondencia de tipos **uno a uno**, entre las subcadenas especiales indicadas en la *cadena de control*, y los demás argumentos pasados en la llamada, que pueden ser constantes, variables o expresiones. Esto es **muy importante**, ya que si no se hace bien, los resultados pueden ser muy diferentes de los esperados.

Según lo visto en la sintaxis de las subcadenas especiales, entre el carácter '%' y el Carácter de Conversión, puede haber, en el orden siguiente, uno o varios de los elementos que a continuación se indican:

- Un signo '-', que indica alineamiento a la izquierda (por defecto, es a la derecha).
- *Anchura*: Un nº entero positivo, que indica la anchura mínima del campo de caracteres. Si el valor que hay que poner en ese campo, ocupa (en caracteres a imprimir) menos que esta anchura indicada, se rellenarán los huecos con espacios en blanco; al inicio del campo, si el alineamiento es a la derecha, y al final si el alineamiento es a la izquierda.
- Un punto que separa la *Anchura* de la *Precisión*.
- *Precisión*: Un número entero positivo, que tiene diferente significado según el tipo de argumento:
  - Cadena de caracteres: nº máximo de caracteres de la misma a escribir. Por defecto, se escriben todos.
  - Nº entero: nº mínimo de cifras a escribir. Si el nº de cifras del número entero a escribir es menor que el indicado por la precisión, se completarían las cifras que faltan mediante 0s a la izquierda. Por defecto, se imprimen todas.
  - Nº de coma flotante: nº de decimales a escribir. Si el nº de decimales del número a imprimir es menor que el indicado por la precisión, se completarían mediante 0s a la derecha. Por defecto, se imprimen 6.

Aparte de caracteres y subcadenas especiales, en la *cadena de control* también pueden aparecer Caracteres o Secuencias de Escape. Con esas Secuencias de Escape, se le indica a *printf* que haga una **impresión especial**. Las Secuencias de Escape son las indicadas en el tema **1.- INTRODUCCIÓN AL LENGUAJE C**, en el punto 1.3.1.2.- Constantes.

Tal y como se ha visto en el prototipo, la función *printf* devuelve un valor *int*. Ese número *int*, representa el nº de caracteres escritos en el buffer de salida, como consecuencia de la llamada. Sin embargo, no se suele guardar ese valor devuelto en ninguna variable (a no ser que se quiera hacer alguna comprobación de cuántos caracteres se han escrito).

Aunque *printf* sea una función estándar, el resultado de su uso varía según el Sistema Operativo. En entornos MS DOS, lo escrito por *printf* en el buffer de salida pasa inmediatamente al dispositivo correspondiente (por defecto, se saca por pantalla), sin que en el fichero fuente se tenga que indicar de ninguna forma. En entornos Linux, sin embargo, el Sistema Operativo, por defecto, espera hasta que en el buffer de salida se haya escrito una línea completa, y es entonces cuando pasa esa línea al dispositivo de salida, quitándola del buffer. Así, para que en un programa para una máquina Linux, la llamada al *printf* suponga que lo escrito en el buffer de salida se pase además inmediatamente al dispositivo correspondiente, hay dos opciones:

- Forzar al Sistema Operativo a que pase lo que hay en el buffer de salida al dispositivo asociado, aunque no sea una línea completa. Para ello, en la misma librería *stdio* hay una función que se encarga de hacer esa

tarea ; se trata de la función *fflush*, y la forma de llamarla para que pase lo que hay en el buffer de salida a la pantalla, es:

```
fflush(stdout);
```

Siendo *stdout* la forma de C de referirse al buffer de salida estándar. Esta sería en realidad la forma estándar de forzar al Sistema Operativo a pasar el contenido del buffer de salida estándar a la pantalla.
Esta opción es en realidad válida para cualquier Sistema Operativo.

- Pasar líneas completas en la *cadena de control*: Terminando la *cadena de control* con el carácter '\n', se pasa una línea completa al buffer de salida y el Sistema Operativo su vez, pasa esa línea automáticamente al dispositivo de salida, posicionando el cursor al principio de la siguiente línea.
  Esta opción es válida en GNU Linux, pero no tiene porqué serlo en otros entornos.

Ejemplo 1:

```
int temperatura=27;

printf("En el año %s la temperatura media fue %d grados\n","2000",temperatura);

printf("En el año %d la temperatura media fue %d grados",2000,temperatura);
fflush(stdout);
```

Ejemplo 2:

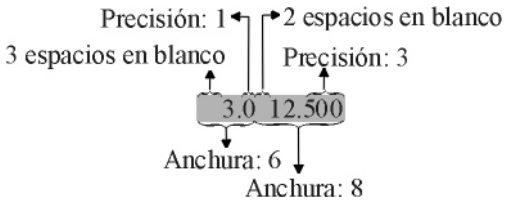
```
printf("Iker tiene %d años\n",34); //En pantalla: Iker tiene 34 años
printf("Iker tiene %2d años\n",34); //En pantalla: Iker tiene 34 años
printf("Iker tiene %4d años\n",34); //En pantalla: Iker tiene   34 años
```

Ejemplo 3:

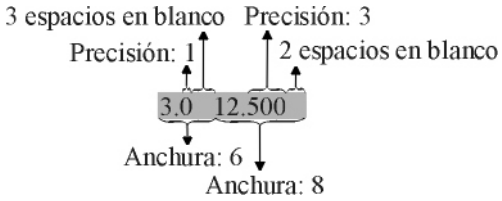
```
int car_imp;

car_imp=printf("\nIker tiene %d años\n",33);
printf("car_imp 1: %d\n",car_imp);

car_imp=printf("\nIker tiene %2d años\n",33);
printf("car_imp 2: %d\n",car_imp);

car_imp=printf("\nIker tiene %5d años\n",33);
printf("car_imp 3: %d\n",car_imp);
```

Ejemplo 4:

```
printf("%6.1f%8.3f\n",3.0,12.5);

/* En pantalla:
```




```
*/

printf("%-6.1f%-8.3f\n",3.0,12.5); //Alineación a la izquierda

/* En pantalla:
```




```
*/
```

3.3.- FUNCIÓN *putchar*

Se trata de una función de salida de datos (como *printf*), que escribe un carácter en salida estándar (asociado, por defecto, a la pantalla). Internamente, funciona también usando el buffer de salida estándar.

Prototipo:

```
int putchar(int c);
```

Escribe en la salida estándar (internamente, en el buffer de salida estándar), el carácter cuyo código ASCII corresponde al número *int* pasado como parámetro. Debido a la conversión implícita, se le puede pasar directamente una variable de tipo *char* (el compilador ya se encargará de hacer la conversión a *int*, antes de ejecutar la función). Devuelve un dato de tipo *int*, que, si todo fue bien, representa el código ASCII del carácter imprimido; si ocurrió algún error en la ejecución, el valor devuelto es la Constante Simbólica EOF.

Para que el Sistema Operativo pase el carácter escrito en el buffer de salida a la pantalla, habrá que llamar a *fflush*, como con *printf*.

Ejemplo:

```
#include <stdio.h>

int main()
{
    char tecla='a';

    printf("\nEscribo 'a' después de : ");
    fflush(stdout);
    putchar(tecla);
    fflush(stdout);

    return 0;
}
```

3.4.- FUNCIÓN *scanf*

La función *scanf* se usa para leer datos de la entrada estándar (que por defecto, está asociado al teclado). Internamente, usa el buffer de entrada estándar para leer las teclas pulsadas. El uso es parecido al de *printf*.

Prototipo:

```
int scanf(<cadena de control>[,<dirección 1>,[...]]);
```

cadena de control: Constante Cadena, en la que se indica, mediante subcadenas especiales (las mismas que *printf*), el formato de lo que se espera que introduzca el usuario desde el teclado. Se escribe como texto entre comillas dobles, y pueden incluirse también Secuencias de Escape.

Los puntos suspensivos indican que puede haber, como en *printf*, cualquier número de argumentos de tipo *dirección* después del primero.

Todo lo que el usuario va introduciendo por el teclado (la entrada estándar por defecto), el Sistema Operativo va guardándolo en el buffer del teclado. Las funciones de entrada de datos como la función *scanf*, leen el contenido de ese buffer para saber qué es lo que ha introducido el usuario en un momento dado. Por lo tanto, lo que se introduce por el teclado no llega directamente al programa que lo está pidiendo, ni siquiera se visualiza en pantalla, sólo se queda almacenado en el buffer del teclado.

El Sistema Operativo, escribe las teclas pulsadas por el usuario en el buffer de entrada. La función *scanf* va leyendo los caracteres que aparecen en el buffer de entrada haciéndolos salir por pantalla, **hasta que en el buffer de entrada** aparezca pulsada la tecla *Return* ('\n'). Así, intenta después interpretar conjuntos de teclas en el buffer de entrada como los valores indicados por las subcadenas especiales de la *cadena de control*, **y mete esos valores obtenidos como contenidos de las direcciones de memoria indicadas después**,

siguiendo el orden en el que aparecen en la llamada. Así, los argumentos que se indican después de la *cadena de control*, no son valores de datos, sino direcciones de memoria. **Hay que asegurarse de que estas direcciones de memoria indicadas son válidas**, es decir, que están en la zona de memoria controlada por el programa.

Las direcciones de memoria controladas por el programa que se conocen hasta este punto del temario, son las direcciones de variables declaradas. Hay también otra forma de trabajar con direcciones de memoria controladas, que es usando *punteros* (tipos de elementos de programación que sirven exclusivamente para guardar valores de direcciones, no de datos).

Así, los argumentos siguientes a la <cadena de control> serán direcciones de variables ya declaradas (indicadas con '**&**' antes del nombre de la variable) o punteros.

Hay que tener cuidado con las cadenas de caracteres o strings. Como ya se verá más adelante, los strings son en realidad *arrays de caracteres*, y todos los arrays, se declaran como punteros. Es decir, que **no hay que poner '&' antes del nombre del string** para ponerlo como argumento en la llamada a *scanf*. Además, las cadenas de caracteres no podrán contener espacios en blanco, tabuladores o '\n', ya que *scanf* está programado para no admitirlos como parte de cadenas de caracteres (en el caso del '\n', scanf lo interpreta como señal de fin de entrada de datos).

Uno de los problemas con los que se encuentra *scanf*, es que el usuario puede introducir lo que quiera desde el teclado, es decir, no tiene porqué hacer caso de lo que le pide el programa (el programa puede pedir un número, y el usuario no hacerle caso e introducir una serie de letras). Así, puede pasar que introduzca datos cuyo formato no es el esperado por *scanf*, porque no es lo que se le indica en la *cadena de control*. Por eso, en cuanto *scanf* lee lo esperado del buffer, o llegue un momento en el que ya no pueda realizar una conversión indicada (el formato introducido no coincide con lo esperado), *scanf* deja de ejecutarse (no intenta obtener más valores del buffer del teclado) y devuelve un número entero (por eso en el prototipo se indica *int* como tipo devuelto) que indica el número de conversiones realizadas con éxito.

Ejemplo:

```
#include <stdio.h>

int main()
{
    int n=0;
    double distancia=0;
    char nombre[20]="";
    int conversiones;

    printf("Valores de las variables y el array, ANTES de la llamada a scanf: \n");
    printf("n: %d\n",n);
```

```
    printf("distancia: %lf\n",distancia);
    printf("nombre: %s\n",nombre);

    printf("\nIntroduzca valores para q los recoja scanf: ");
    fflush(stdout);
    conversiones=scanf("%d%lf%s",&n,&distancia,nombre);

    printf("\nValores de las variables y el array, DESPUES de la llamada a scanf:\n");
    printf("n: %d\n",n);
    printf("distancia: %lf\n",distancia);
    printf("nombre: %s\n",nombre);
    printf("Conversiones realizadas con éxito: %d\n",conversiones);

    return 0;
}
```

/* Distintos funcionamientos de *scanf* según lo introducido por el usuario:

Caso 1

```
Valores de las variables y el array, antes de la llamada al scanf():
n: 0
distancia: 0.000000
nombre:

Introduzca valores para que los recoja scanf(): 123456789.123456asdgadgfraerg

Valores de las variables y el array, despuús de la llamada al scanf():
n: 123456789
distancia: 0.123456
nombre: asdgadgfraerg
Conversiones realizadas con Úxito: 3
```

Caso 2

```
Valores de las variables y el array, antes de la llamada al scanf():
n: 0
distancia: 0.000000
nombre:

Introduzca valores para que los recoja scanf(): 123456789.123asfgasgasg

Valores de las variables y el array, despuús de la llamada al scanf():
n: 123456789
distancia: 0.123000
nombre: asfgasgasg
Conversiones realizadas con Úxito: 3
```

Caso 3

```
Valores de las variables y el array, antes de la llamada al scanf():
n: 0
distancia: 0.000000
nombre:

Introduzca valores para que los recoja scanf(): 123456789.52689456312sdfasdg

Valores de las variables y el array, despuús de la llamada al scanf():
n: 123456789
distancia: 0.526895
nombre: sdfasdg
Conversiones realizadas con Úxito: 3
```

Caso 4

```
Valores de las variables y el array, antes de la llamada al scanf():
n: 0
distancia: 0.000000
nombre:

Introduzca valores para que los recoja scanf(): 123456789.zxdgasgfadg

Valores de las variables y el array, despuús de la llamada al scanf():
n: 123456789
distancia: 0.000000
nombre:
Conversiones realizadas con Úxito: 1
```

Caso 5

```
Valores de las variables y el array, antes de la llamada al scanf():
n: 0
distancia: 0.000000
nombre:

Introduzca valores para que los recoja scanf(): asdgadfgasdfh

Valores de las variables y el array, despuús de la llamada al scanf():
n: 0
distancia: 0.000000
nombre:
Conversiones realizadas con Úxito: 0
```

*/

En la práctica, hay que tener en cuenta dos cosas:

- Aunque en un mismo *scanf* se puedan recoger valores para más de una variable, como lo introducido por el usuario no es fiable (el usuario no tiene porqué hacer lo que el programador quiere que haga), lo recomendado es recoger sólo un valor con cada *scanf*, y si tiene que pertenecer a un cierto rango de valores, comprobar que cumple esa condición, antes de seguir ejecutando el programa. El algoritmo a seguir suele ser el mostrado en la siguiente figura:

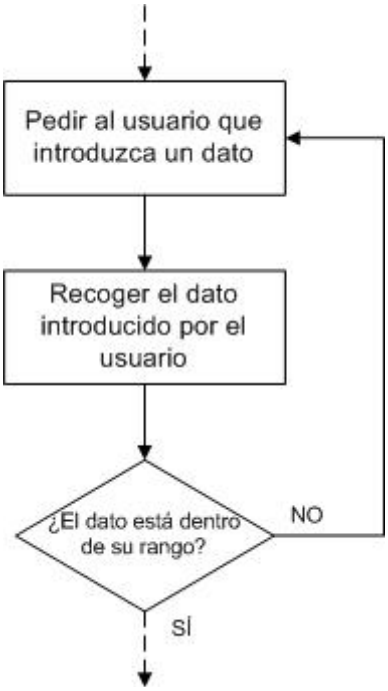


Figura 1: Algoritmo de petición de datos

Para llevar a cabo lo indicado por el algoritmo anterior, se han de usar estructuras de control, que se verán más adelante.

- **Sólo** los **caracteres** introducidos por el usuario con los que se pudieron obtener los formatos de valores esperados (se pudo realizar una **conversión adecuada**), **desaparecen** del buffer del teclado. Así, si una llamada a *scanf* no pudo terminarse con todos datos que se esperaban recoger, hay que tener en cuenta, que **puede haber en el buffer del teclado** caracteres que no hayan desaparecido. Esos **caracteres residuales** pueden dar problemas en la siguiente llamada a una función de entrada de datos, ya que, si esa función no *limpia* (borra) el buffer antes de empezar a leer de él, leería esos caracteres residuales de la llamada anterior, en vez esperar a que el usuario introduzca nuevos caracteres. La función *scanf* es una de esas funciones de entrada que no limpian el buffer de entrada antes de empezar a leer de él, con lo cuál, puede tener problemas con caracteres residuales.

  Para resolver ese problema, se llama a una función que limpia el buffer del teclado. La función que se usa es *__fpurge* (hay que incluir fichero de cabecera stdio_ext.h), y la llamada que se hace es la siguiente:

  ```
  __fpurge(stdin);
  <llamada a una función de Entrada que no hace limpieza de buffer de entrada>
  ```

  El argumento actual *stdin* es la forma en C de referirse al buffer de la entrada estándar (el teclado).

Ejemplo:

```
/* Modificamos el ejemplo anterior para ver el funcionamiento de scanf ante
caracteres residuales */

#include <stdio.h>
#include <stdio_ext.h>

int main()
{
    int n=0;
    double distancia=0;
    char nombre[20]="";
    int conversiones;

    printf("\n\nIntroduzca valores para q los recoja scanf: ");
    fflush(stdout);
    conversiones=scanf("%d%lf%s",&n,&distancia,nombre);

    printf("\nValores de las variables y el array, DESPUES de la llamada a scanf: \n");
    printf("n: %d\n",n);
    printf("distancia: %lf\n",distancia);
    printf("nombre: %s\n",nombre);
    printf("Conversiones realizadas con éxito: %d\n",conversiones);

    printf("\nCogemos los 3 primeros caracteres residuales de la llamada anterior:");
    fflush(stdout);
    scanf("%3s",nombre);
    // Si realmente hay caracteres residuales, scanf no espera a nada
    printf("\nnombre: %s\n",nombre);

    printf("\n\nBorramos el buffer y esperamos a que el usuario meta un string:");
    fflush(stdout);
    __fpurge(stdin);
    scanf("%s",nombre);
    //Ahora sí espera, pq con fflush, el buffer de entrada se ha vaciado
    printf("\nnombre: %s\n",nombre);

    return 0 ;
}

/* Resultado en pantalla, cuando el usuario introduce una cadena con  un carácter
de tabulación:

Introduzca valores para q los recoja scanf: 123.8362asdfgh      qwertyu

Valores de las variables y el array, DESPUES de la llamada a scanf:
n: 123
distancia: 0.836200
nombre: asdfgh
Conversiones realizadas con úxito: 3
Cogemos los 3 primeros caracteres residuales de la llamada anterior:
nombre: qwe

Borramos el buffer y esperamos a q el usuario meta un string: xzvcvxcbx

nombre: xzvcvxcbx

*/
```

3.5.- FUNCIÓN *getchar*

Como la función *scanf*, *getchar* es una función de entrada de datos. Así, lo que hace, es leer de la entrada estándar, (por defecto, asociado al teclado) usando internamente un buffer para ello.

Prototipo:

```
int getchar();
```

Devuelve un dato de tipo *int* (4 bytes), que en realidad, representa el código ASCII de algún carácter introducido por el usuario por teclado.

Espera a que el Sistema Operativo escriba en el buffer del teclado **un carácter seguido de** la pulsación de la tecla ***Return***, los imprime por pantalla (a través del buffer de salida) y devuelve, en formato int, su código ASCII. Si hubiera más de un carácter antes de la tecla *Return*, los imprimiría todos pero sólo devolvería el código del primer carácter introducido. Todo lo que hubiera en el buffer hasta *Return*, desaparecerá del buffer de entrada (y del de salida, ya que el Sistema Operativo lo va imprimiendo en pantalla).

Ejemplo:

```
#include <stdio.h>

int main()
{
    char tecla;

    printf("\nIntroduzca una tecla seguida de Return: ");
    fflush(stdout);
    tecla=getchar();

    printf("\nTecla pulsada: %c\n\n",tecla);

    return 0;
}

/* Distintos funcionamientos de getchar según lo introducido por el usuario:

Caso 1

Introduzca una tecla seguida de Return: a

Tecla pulsada: a

Caso 2

Introduzca una tecla seguida de Return: EJKLDFGJKLDFG

Tecla pulsada: E

*/
```

La función *getchar* tampoco limpia el buffer de entrada antes de leer de él. Por eso, hay que tener cuidado de que no haya caracteres residuales en el mismo, y llamar a *__fpurge* cuando sea necesario.

Ejemplo:

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

int main()
{
    int n=0;
    double distancia=4;
    char nombre[20]="";
    int conversiones;
    char c;

    printf("Valores de las variables y el array, ANTES de la llamada a scanf: \n");
    printf("n: %d\n",n);
    printf("distancia: %lf\n",distancia);
    printf("nombre: %s\n",nombre);

    printf("\nIntroduzca valores para q los recoja scanf: ");
    fflush(stdout);
    conversiones=scanf("%d%lf%s",&n,&distancia,nombre);

    printf("\nValores de las variables y el array, DESPUES de la llamada a scanf: \n");
    printf("n: %d\n",n);
    printf("distancia: %lf\n",distancia);
    printf("nombre: %s\n",nombre);
    printf("Conversiones realizadas con éxito: %d\n",conversiones);

    //__fpurge(stdin); Para que todo fuera bien, habría que limpiar el buffer de entrada

    c=getchar();
    printf("\n\nc: %c\n",c);

    return 0;
}
```

/* Resultado en pantalla, cuando el usuario introduce valores con los que la primera llamada a *scanf* no puede realizar todas las conversiones:

```
Valores de las variables y el array, ANTES de la llamada a scanf:
n: 0
distancia: 4.000000
nombre:

Introduzca valores para q los recoja scanf: 123.qwtqwet

Valores de las variables y el array, DESPUES de la llamada a scanf:
n: 123
distancia: 4.000000
nombre:
Conversiones realizadas con Úxito: 1

c: q
```

*/