

# Entrada y Salida en C

En general, en el mundo de la informática la entrada y salida de datos es un tema complejo. Desafortunadamente la entrada y salida en C no es una excepción. En estos apuntes explicaremos algunos conceptos básicos para que el lector pueda leer y escribir datos de tipo numérico y carácter en C. Para profundizar más en este tema deberá consultar otras fuentes.

Para leer datos utilizaremos la función *scanf* y para escribir datos la función *printf*. Ambas funciones son parte de la biblioteca estándar de C, y para utilizarlas es preciso incluir el fichero de cabecera *stdio.h*.

## 1. Lectura de datos

Para leer un dato utilizaremos la función *scanf* de la siguiente forma: `scanf ("%X", &var);`. La *X* representa el tipo de los datos que queremos leer y debe ser sustituida por (hay más posibilidades que las enumeradas aquí):

- `d`, para leer un entero (`int`).
- `f`, para leer un valor de punto flotante (`float`).
- `lg`, para leer un valor de punto flotante en doble precisión (`double`).
- `c`, para leer un carácter (`char`).

*var* es el nombre de la variable donde se almacenará el valor leído. Observe que el nombre de la variable va precedido del operador `&`. Durante el curso verá por qué es necesario poner este operador en la función *scanf*.

## 2. Escritura de datos

La función *printf* nos permite escribir una secuencia de caracteres en la salida, su formato es: `printf ("Secuencia a escribir" [, expresión1, expresión2, ...]);`. La serie de expresiones es opcional. Por ejemplo: para escribir *Hola* en la salida utilizaremos `printf ("Hola");`.

Para producir un salto de línea en la salida se utiliza la secuencia `\n`. Por ejemplo, `printf ("Hola\n\na todos\n");` escribe *Hola*, una línea vacía y después *a todos* seguido de un salto de línea.

A veces queremos escribir en la salida el valor almacenado por una variable *var*, para ello hay que realizar una conversión del formato numérico binario en que se almacena la variable a una secuencia de caracteres. Con *printf* esto se hace así: `printf ("%X", var);`, donde *X* especifica el tipo de la variable *var*, y sus posibles valores son los mismos que los especificados para *scanf*.

Realmente con *printf* podemos especificar una secuencia de caracteres que incluye conversiones de formato. Por ejemplo, `printf ("El valor de x es %d\n", x);`, escribiría *El valor de x es 8* seguido de un salto de línea —suponiendo que la variable *x* es de tipo entero y almacena el valor 8.

Se puede hacer más de una conversión de formato con *printf*. Por ejemplo: `printf ("La base y altura del triángulo es: %f y %f\n", base, altura)` mostrará en la salida *La base y altura del triángulo es: b y a* donde *b*

y *a* representan los valores almacenados por las variables *base* y *altura* respectivamente. Por cada conversión de formato que aparezca en la secuencia de caracteres debe haber una variable del tipo correspondiente en la lista de expresiones. La primera conversión se aplica a la primera expresión de la lista, la segunda conversión a la segunda expresión de la lista, y así sucesivamente.

Para terminar con *printf* decir que las conversiones de formato se aplican realmente a expresiones y no a variables —una variable es un tipo particular de expresión. Por lo tanto, lo siguiente es válido:

```
printf ("El área del triángulo es: %f\n", base * altura / 2);
```

y escribirá El área del triángulo es: *x* seguido de un salto de línea, donde *x* representa el resultado de evaluar la expresión  $\frac{base*altura}{2}$ .

## 2.1. Formato de las conversiones de formato

Hemos visto que para realizar una conversión de formato con *printf* hacíamos: `printf ("%X", expresión);`. En esta sección veremos que podemos dar cierto formato a la secuencia de caracteres que produce la conversión. Concretamente podemos utilizar *printf* así: `printf ("%-Anchura.PrecisiónX", expresión);`. Tanto *-* como *Anchura* y *Precisión* son opcionales e indican lo siguiente:

- *-*, tiene sentido si se utiliza la también la opción de *Anchura*. Por defecto, se justifica a la derecha la salida de la conversión. Si especificamos *-* se justificará a la izquierda.
- *Anchura*, es un número que indica la anchura en caracteres que como mínimo tiene que producir la salida del número —si la anchura es mayor que el número de dígitos de un número se rellena con espacios en blanco.
- *Precisión*, tiene sentido para números reales, indica el número de dígitos de la parte fraccionaria. Si se pone sólo punto o un número negativo se entiende que es cero.

Por ejemplo, `printf ("%8.2f", 4.756);` escribe cuatro espacios en blanco, seguidos de la secuencia de caracteres 4.76.

## 3. Algunos comentarios sobre *scanf*

En esta sección se describen algunas peculiaridades de la lectura de datos con *scanf*. En concreto queremos destacar los siguientes aspectos:

- Cuando el usuario introduce un dato desde el teclado, hasta que no pulsa la tecla *Enter* los datos tecleados no están disponibles para el programa, y por tanto, el programa no leerá nada hasta que el usuario no pulse *Enter*. Los datos tecleados cuando se pulsa *Enter* se almacenan en un *buffer* de lectura. De este *buffer* es de donde el programa leerá los datos.
- Cuando se lee un dato de tipo numérico con *scanf* ocurre lo siguiente. Primero se descartan todos los caracteres blancos que haya en el *buffer* —por caracteres blancos se entiende el espacio en blanco, el tabulador y el salto de línea—, después se lee del *buffer* el número introducido hasta que se encuentre un carácter que no sea de tipo numérico —normalmente, un

espacio blanco. La próxima lectura con *scanf* comenzará con este último carácter.

- Cuando se lee un dato de tipo carácter se lee el siguiente carácter que haya en el *buffer* —sea un carácter blanco o no. Es decir, no hay descarte de caracteres blancos.
- *scanf* es una función, y devuelve el número de lecturas que realizó con éxito. Como nosotros utilizamos *scanf* para leer un único dato *scanf* devolverá 0 ó 1. 0 si no consiguió leer con éxito y 1 si leyó con éxito.

### 3.1. El problema de los caracteres no procesados en el *buffer*.

A continuación exponemos dos problemas que surgen al leer datos con *scanf* y una posible solución a ellos. Para ejemplificar el primer problema suponga el siguiente programa:

```
#include <stdio.h>

int main ()
{
    int x, y;
    printf ("Introduzca un entero: ");
    scanf ("%d", &x);
    printf ("Introduzca un segundo entero: ");
    scanf ("%d", &y);
    printf ("Los valores introducidos son: %d y %d\n", x, y);
}
```

Si el usuario teclea un número, pulsa *Enter* y después otro número y otra vez *Enter* entonces el programa tendrá el comportamiento “esperado”. Ejecute el programa de nuevo y cuando se pida el primer entero escriba ‘2 3’ y pulse *Enter*. Observe el resultado de la ejecución. Lo que ocurre es lo siguiente. Al realizar el primer *scanf* el programa no lee datos hasta que el usuario no pulse *Enter*. Esta vez el usuario ha tecleado dos números en una línea —separados por espacios en blanco. De esta forma, cuando se realiza el segundo *scanf* se empieza a leer a partir del primer espacio en blanco almacenado en el *buffer* — en el *buffer* quedan caracteres aún no procesados. Si usted quiere leer un único dato por línea, independientemente de que el usuario teclee más de un dato en una línea puede escribir lo siguiente.

```
#include <stdio.h>

int main ()
{
    int x, y;
    printf ("Introduzca un entero: ");
    scanf ("%d", &x);
    while (getchar () != '\n');
    printf ("Introduzca un segundo entero: ");
```

```

scanf ("%d", &y);
while (getchar () != '\n');
printf ("Los valores introducidos son: %d y %d\n", x, y);
}

```

La instrucción situada tras el *scanf* elimina —realmente lee, aunque no haga nada con ellos— todos los caracteres pendientes de procesamiento almacenados en el *buffer* —incluido el *Enter*. Por lo tanto, tras leerse el primer número se eliminará todo lo que el usuario haya tecleado en el resto de la línea.

A continuación describimos un problema muy parecido al anterior y que tiene la misma solución:

```

#include <stdio.h>

int main ()
{
    int x;
    char c;
    printf ("Introduzca un entero: ");
    scanf ("%d", &x);
    printf ("Introduzca un carácter: ");
    scanf ("%c", &c);
    printf ("Los valores introducidos son: %d y %c\n", x, c);
}

```

Este programa lee un entero y después un carácter. Ejecútelo y teclee un número entero seguido de *Enter*. Observe la salida. ¿Qué ha ocurrido? Lo siguiente. Hasta que el usuario no pulsa *Enter* el programa no dispone de los datos. Entonces, *scanf* lee el número y detiene la lectura al encontrar el *Enter*. Sin embargo, este *Enter* queda en el *buffer* pendiente de procesamiento. A continuación se ejecuta el segundo *scanf*. Se empieza leyendo por lo que queda en el *buffer* —el *Enter*. Como estamos leyendo un carácter el *Enter* no se descarta y se almacena en la variable *c*. Para evitar esto se recomienda la misma solución que para el problema anterior. Es decir, utilizar `while (getchar () != '\n');` para eliminar los caracteres que quedan por procesar de la línea en el *buffer* —en este caso el *Enter*. La solución sería:

```

#include <stdio.h>

int main ()
{
    int x;
    char c;
    printf ("Introduzca un entero: ");
    scanf ("%d", &x);
    while (getchar () != '\n');
    printf ("Introduzca un carácter: ");
    scanf ("%c", &c);
    while (getchar () != '\n');
    printf ("Los valores introducidos son: %d y %c\n", x, c);
}

```