Unidad 1 – E/S

Índice

La librería stdio.h contiene un conjunto de funciones que nos permiten leer datos desde teclado y/o enviarlos a pantalla.

stdio.h, que significa "standard input-output header" (cabecera estándar E/S), es el archivo de cabecera que contiene las definiciones de las macros, las constantes, las declaraciones de funciones de la biblioteca estándar del lenguaje de programación C para hacer operaciones, estándar, de entrada y salida, así como la definición de tipos necesarias para dichas operaciones.

En C y sus derivados, todas las funciones son declaradas en archivos de cabecera. Así, los programadores tienen que incluir el archivo de cabecera stdio.h dentro del código fuente para poder utilizar las funciones que están declaradas.

```
#include <stdio.h>
int main()
{
/* my first program in C */
printf("Hello, World! \n");
return 0;
}
```

Las funciones declaradas en stdio.h pueden clasificarse en dos categorías: funciones de manipulación de ficheros y funciones de manipulación de entradas y salidas.

Nombre	Descripción
Funciones de manipulación de ficheros	
`fclose`	Cierra un fichero a través de su puntero.
`fopen`, `freopen`, `fdopen`	Abre un fichero para lectura, para escritura/reescritura o para adición.
`remove`	Elimina un fichero.
`rename`	Cambia al fichero de nombre.
`rewind`	Coloca el indicador de posición de fichero para el stream apuntado por stream al comienzo del fichero.
`tmpfile`	Crea y abre un fichero temporal que es borrado cuando cerramos con la función *fclose()*.

Funciones de manipulación de entradas y salidas.	
`clearerr`	Despeja los indicadores de final de fichero y de posición de fichero para el stream apuntado por stream al comienzo del fichero.
`feof`	Comprueba el indicador de final de fichero.
`ferror`	Comprueba el indicador de errores.
`fflush`	Si stream apunta a un *stream* de salida o de actualización cuya operación más reciente no era de entrada, la función *fflush* envía cualquier dato aún sin escribir al entorno local o a ser escrito en el fichero; si no, entonces el comportamiento no está definido. Si stream es un puntero nulo, la función fflush realiza el despeje para todos los *streams* cuyo comportamiento está descrito anteriormente.
`fgetpos`	Devuelve la posición actual del fichero.
`fgetc`	Devuelve un carácter de un fichero.
`fgets`	Consigue una cadena de caracteres de un fichero.

La lista de funciones de E/S es mucho más larga de forma que sólo se muestra aquí una parte de esta lista.

A continuación, vamos a ir analizando algunas de ellas.

1.- La función printf

La función printf (que deriva su nombre de “print formatted”) imprime un mensaje por pantalla utilizando una “cadena de formato” que incluye las instrucciones para mezclar múltiples cadenas en la cadena final a mostrar por pantalla. Lenguajes como Java también incluyen funciones similares a esta

printf es una función especial porque recibe un número variable de parámetros. El primer parámetro es fijo y es la cadena de formato. En ella se incluye texto a imprimir literalmente y marcas a reemplazar por texto que se obtiene de los parámetros adicionales. Por tanto, printf se llama con tantos parámetros como marcas haya en la cadena de formato más uno (la propia cadena de formato). El siguiente ejemplo muestra cómo se imprime el valor de la variable contador.

```
printf("El valor es %d.\n", contador);
```

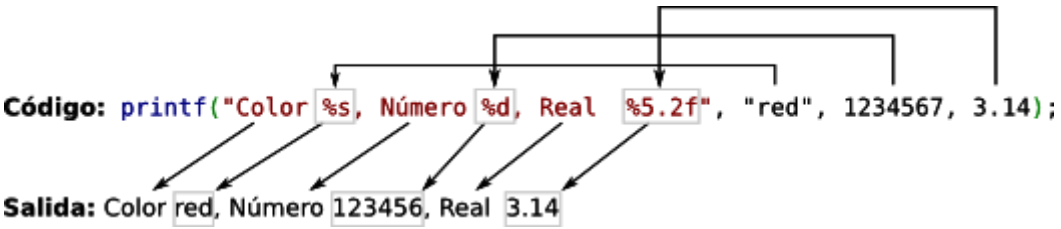
El símbolo “%” denota el comienzo de la marca de formato. La marca “%d” se reemplaza por el valor de la variable contador y se imprime la cadena resultante. El símbolo “\n” representa un salto de línea. La salida, por defecto, se justifica a

la derecha del ancho total que le hallamos dado al campo, que por defecto tiene como longitud la longitud de la cadena.

```
#include <stdio.h>

int main()
{
/* my first program in C */
for(int i=0;i<10;i++)
  {
     printf("Valor de i %d, .\n", i); }
return 0;
}
```

Si en la cadena de formato aparecen varias marcas, los valores a incluir se toman en el mismo orden en el que aparecen. La siguiente figura muestra un ejemplo en el que la cadena de formato tiene tres marcas, %s, %d y %5.2f, que se procesan utilizando respectivamente la cadena “red”, el entero 1234567 y el número real 3.14.



No se comprueba que el número de marcas en la cadena de formato y el número de parámetros restantes sea consistente. En caso de error, el comportamiento de printf es indeterminado.

Las marcas en la cadena de formato deben tener la siguiente estructura (los campos entre corchetes son optativos):

%[parameter][flags][width][.precision][length]type

Toda marca, por tanto, comienza por el símbolo “%” y termina con su tipo. Cada uno de los nombres (parameter, flags, width, precision, length y type) representa un conjunto de valores posibles que se explican a continuación.

Parameter	Descripción
n$	Se reemplaza “n” por un número para cambiar el orden en el que se procesan los argumentos. Por ejemplo `%3$d` se refiere al tercer argumento independientemente del lugar que ocupa en la cadena de formato.

Por ejemplo, en el siguiente código

printf("Valor de i %2$d, %d.\n", i, 2*i);

primero escribe el producto 2*i y después i aunque el orden en el que aparecen en la lista sea distinto.

Flags	Descripción
número	Rellena con espacios (o con ceros, ver siguiente flag) a la izquierda hasta el valor del número.
0	Se rellena con ceros a la izquierda hasta el valor dado por el flag anterior. Por ejemplo "`%03d`" imprime un número justificado con ceros hasta tres dígitos.
+	Imprimir el signo de un número
-	Justifica el campo a la izquierda (por defecto ya hemos dicho que se justifica a la derecha)
#	Formato alternativo. Para reales se dejan ceros al final y se imprime siempre la coma. Para números que no están en base 10, se añade un prefijo denotando la base.

8.3.4. La función

Por ejemplo, el siguiente código

```
printf("Valor de i %04d, %d.\n", i, 2*i);
```

mostraría

```
Valor de i 0000, 0.
Valor de i 0001, 2.
Valor de i 0002, 4.
Valor de i 0003, 6.
```

De la misma forma, el código

```
printf("Valor de i %04d, %+d.\n", i, 2*i);
```

Mostraría como resultado

```
Valor de i 0000, +0.
Valor de i 0001, +2.
Valor de i 0002, +4.
Valor de i 0003, +6.
```

Width	Descripción
número	Tamaño del ancho del campo donde se imprimirá el valor.
*	Igual que el caso anterior, pero el número a utilizar se pasa como parámetro justo antes del valor. Por ejemplo `printf("%*d", 5, 10)` imprime el número 10, pero con un ancho de cinco dígitos (es decir, rellenará con 3 espacios en blanco a la izquierda).

Para asignar un ancho en la representación de una cadena utilizamos el parámetro width. Asi, el siguiente código

```
printf("Valor de i %04d, %7d.\n", i, 2*i);
```

mostraría

Valor de i 0000, 0.
Valor de i 0001, 2.
Valor de i 0002, 4.

En lugar de un valor de ancho podemos dejarlo como variable e indicarlo en la cadena de valores justo antes del valor que queremos formatear. Por ejemplo

printf("Valor de i %04d, %*d.\n", i, 4,2*i);

Obtendríamos

Valor de i 0000, 0.
Valor de i 0001, 2.
Valor de i 0002, 4.

Precision	Descripción
número	Tamaño de la parte decimal para números reales. Número de caracteres a imprimir para cadenas de texto
*	Igual que el caso anterior, pero el número a utilizar se pasa como parámetro justo antes del valor. Por ejemplo `printf("%.*s", 3, "abcdef")` imprime "abc".

Los formateadores son

Formateador	Salida
%d ó %i	entero en base 10 con signo (int) printf ("el numero enteronen base 10 es: %d" , -10);
%u	entero en base 10 sin signo (int)
%o	entero en base 8 sin signo (int)
%x	entero en base 16, letras en minúscula (int)
%X	entero en base 16, letras en mayúscula (int)
%f	Coma flotante decimal de precisión simple (float)
%lf	Coma flotante decimal de precisión doble (double)
%ld	Entero de 32 bits (long)
%lu	Entero sin signo de 32 bits (unsigned long)
%e	La notación científica (mantisa / exponente), minúsculas (decimal precisión simple o doble)
%E	La notación científica (mantisa / exponente), mayúsculas (decimal precisión simple o doble)
%c	carácter (char)
%s	cadena de caracteres (string)

2.- Funciones de entrada

2.1 La función gets

Para leer una línea completa contamos con diferentes funciones; una de ellas es gets:

```
#include <stdio.h>
char *gets(char *s);
```

Los caracteres leídos de entrada se guardan en el array s. La función deja de leer y añade el carácter de terminación '\0' cuando encuentra el carácter de nueva línea o el de fin de fichero EOF. Si todo va bien devuelve s, y si hay algún error devuelve un puntero a NULL.

```
#include <stdio.h>
int main()
{
char s1[30];
gets(s1);
printf("%s",s1);
return 0;
}
```

Ahora bien, gets es una función insegura tal como lo indica el compilador (warning: this program uses gets(), which is unsafe.). El problema es que no puedes controlar el número de caracteres que introduce el usuario pudiendo ocurrir que se copien en la cadena más caracteres que los permitidos por su tamaño máximo.

2.2 La función Scanf

scanf es la más versátil de las tres dado que puede leer distintos tipos de datos (cadenas, enteros, reales…) según el formato especificado. En cambio, gets y fgets sólo leen cadenas de caracteres (nótese la ‘s’ final del nombre que hace referencia en este caso a string).

En este ejemplo se lee un número entero desde teclado.

```
#include <stdio.h>
int main()
{
int i;
scanf("%d", &i);
printf("%d",i);
return 0;
}
```

Ambas funciones gets y scanf usadas para leer cadenas de caracteres son propensas a errores, pues no controlan que el usuario haya introducido más caracteres que el que puede albergar el array definido para almacenarlos. Para ello, contamos con dos funciones que hacen segura la lectura de cadenas introducidas por el usuario, que son fgets y getline.

2.3. La función fgets

La alternativa segura de gets es fgets que si permite establecer el máximo de caracteres que pueden leerse. Un ejemplo de uso sería:

La sintaxis de la función fgets es:

```
#include <stdio.h>
char *fgets(char *s, int n, FILE *stream);
```

y un ejemplo sería

```
char s[30];
fgets(s, 30, stdin);
```

s referencia a un array de caracteres, que almacena lo que lee de un fichero apuntado por stream. n especifica el número máximo de elementos del array (caracteres). La función lee hasta n-1 caracteres, y añade el caracter nulo para que la cadena quede bien formada. Si no ha llegado hasta n-1 caracteres, pero llega a un carácter de nueva línea, '\n', también para de leer y devuelve la cadena (con ese carácter de nueva línea incluido). Si no hay errores, la función devuelve el puntero s. Si encuentra el final de fichero (EOF) o existe algún error, devuelve null.

En este ejemplo, se lee desde teclado una cadena hasta que se pulse enter o se completen los 30 caracteres indicados (en realidad son 29 ya que se añade el final de cadena).

fgets es así más segura que gets, pero tienes que lidiar con un par de cosas. Una es que si has introducido más caracteres de los que ha podido leer, el resto de caracteres se quedan en el buffer de entrada (en stdin), así que tendrás que eliminarlos para que no sean leídos en el próximo fgets si no quieres. Lo segundo es que fgets incluye el carácter '\n' al final, así que tendrás que quitarlo.