8.3.4. 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 (ver Método printf de la clase PrintStream).

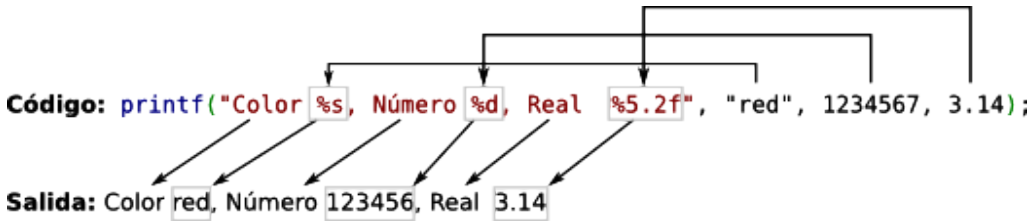
printf es una función especial porque recibe un número variable de parámetros. El primer parámetro es fijo y es la cadena de formato. En ella se incluye texto a imprimir literalmente y *marcas* a reemplazar por texto que se obtiene de los parámetros adicionales. Por tanto, printf se llama con tantos 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.

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.
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.
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).
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”.
Length	Descripción
hh	Convertir variable de tipo `char` a entero e imprimir
h	Convertir variable de tipo `short` a entero e imprimir
l	Para enteros, se espera una variable de tipo `long`.
ll	Para enteros, se espera una variable de tipo `long long`.
L	Para reales, se espera una variable de tipo `long double`.
z	Para enteros, se espera un argumento de tipo `size_t`.
Type	Descripción
%c	Imprime el carácter ASCII correspondiente

%d, %i	Conversión decimal con signo de un entero
%x, %X	Conversión hexadecimal sin signo
%p	Dirección de memoria (puntero)
%e, %E	Conversión a coma flotante con signo en notación científica
%f, %F	Conversión a coma flotante con signo, usando punto decimal
%g, %G	Conversión a coma flotante, usando la notación que requiera menor espacio
%o	Conversión octal sin signo de un entero
%u	Conversión decimal sin signo de un entero
%s	Cadena de caracteres (terminada en '\0')
%%	Imprime el símbolo %

Ejemplos de marcas de formato

Las marcas de formato que se incluyen como parte de la cadena que se pasa como primer parámetro a `printf` ofrece muchas posibilidades. A continuación se muestran algunos ejemplos:

- Especificar el ancho mínimo: Podemos poner un entero entre el símbolo de porcentaje (%) y el especificador de formato, para indicar que la salida alcance un ancho mínimo. Por ejemplo, `%10f` asegura que la salida va a tener al menos `10` espacios de ancho. Esto es útil cuando se van a imprimir datos en forma de columnas. Por ejemplo, si tenemos:

```
int num = 12;
int num2 = 12345;
printf("%d\n",num2);
printf("%5d\n",num);
```

se imprime:

```
12345
   12
```

- Alinear la salida: Por defecto, la salida cuando se especifica ancho mínimo está justificada a la derecha. Para justificarla a la izquierda, hay que preceder el dígito de la anchura con el signo menos (-). Por ejemplo, `%-12d` especifica un ancho mínimo de 12, saca la salida justificada a la izquierda.

- Especificador de precisión: Puedes poner un punto (.) y un entero después de especificar un ancho de campo mínimo para especificar la precisión. En un dato de tipo float, esto permite especificar el número de decimales a sacar. En un dato de tipo entero o en cadenas de caracteres, especifica el ancho o longitud máxima.

Comprueba con estas preguntas si has entendido este documento.

1. Queremos imprimir la cadena guardada en `char string[30];` con la función `printf`:
    - Tenemos que usar `printf("%c\n",string[0]);`.

- Tenemos que usar `printf("%s\n",string[0]);`.
- Tenemos que usar `printf("%s\n",string);`

Corregir

8.3.3. La función scanf

8.4. Funciones de entrada para leer strings de manera segura