9

Entradas/salidas con formato

Objetivos

- Comprender flujos de entrada y de salida.
- Ser capaz de utilizar todas las capacidades de formato de impresión.
- Ser capaz de utilizar todas las capacidades de formato de entrada.

Todas las noticias posibles de imprimir.
Adolph S. Ochs

¿Qué loca quimera? ¿Qué lucha por escapar?
John Keats

No quiten las mojoneras en los límites de los campos.
Amenemope

El fin debe de justificar los medios.
Matthew Prior

Sinopsis

Resumen • Terminología • Errores comunes de programación • Prácticas sanas de programación • Sugerencias de portabilidad • Ejercicios de autoevaluación • Respuestas a los ejercicios de autoevaluación • Ejercicios.

9.1 Introducción

Una parte importante de la solución de cualquier problema es la presentación de los resultados. En este capítulo analizamos a fondo las características de formato de **printf** y de **scanf**. Estas funciones introducen *flujos de datos estándar de entrada* y extraen *flujos de datos estándar de salida*, respectivamente. Las otras cuatro funciones que utilizan entradas y salidas estándar **gets**, **puts**, **getchar**, y **putchar** fueron analizadas en el capítulo 8. Incluya el archivo de cabecera **<stdio.h>** en aquellos programas que llamen a estas funciones.

Muchas características de **printf** y de **scanf** han sido ya analizadas en el texto. Este capítulo resume estas características y presenta muchas otras. En el capítulo 11 se analizarán varias otras funciones incluidas en la biblioteca estándar de entrada/salidas (**stdio**).

9.2 Flujos

Toda entrada y salida se ejecuta mediante *flujos* —secuencias de caracteres organizadas en líneas. Cada línea está formada de cero o más caracteres y está terminada por el carácter de nueva línea. El estándar indica que las aplicaciones de ANSI C deben soportar líneas de por lo menos 254 caracteres, incluyendo un carácter de nueva línea de terminación.

Cuando empieza la ejecución del programa, de forma automática se conectan tres flujos al programa. Por lo regular, el flujo estándar de entrada se conecta al teclado y el flujo estándar de salida se conecta a la pantalla. Con frecuencia los sistemas operativos permiten que estos flujos sean *redirigidos* a otros dispositivos. Un tercer flujo, el *error estándar*, se conecta a la pantalla.

Los mensajes de error son extraídos o desplegados al flujo estándar de error. Los flujos se analizan en detalle en el capítulo 11, "Procesamiento de archivos".

9.3 Salida con formato utilizando printf

Utilizando **printf** se consigue un formato de salida preciso. Cada llamada **printf** contiene una *cadena de control de formato*, que describe el formato de la salida. La cadena de control de formato consiste de *especificadores de conversión, banderas, anchos de campo, precisiones y caracteres literales.* Junto con el signo de por ciento (%), éstos forman *especificaciones de conversión*. La función **printf** puede llevar a cabo las siguientes capacidades de formato, cada una de las cuales se analiza en este capítulo.

1. *Redondear* valores de punto flotante, a un número indicado de valores decimales.
2. *Alinear* una columna de números, con puntos decimales apareciendo uno por encima del otro.
3. *Salidas justificadas* a la *derecha* o a la *izquierda*.
4. *Insertar caracteres literales* en posiciones precisas en una línea de salida.
5. Representación en formato exponencial de números de punto flotante.
6. Representación en formato octal y hexadecimal de enteros no signados. Vea el apéndice E, "Sistemas numéricos" para más información sobre valores octales y hexadecimales.
7. Despliegue de todo tipo de datos con anchos de campo de tamaño fijo y precisiones.

La función **printf** tiene la forma:

printf (*format-control-string*, *other-arguments*) ;

Format-control-string describe el formato de salida, y *other-arguments* (estos son opcionales) corresponden a cada especificación de conversión existente en *format-control-string*. Cada especificación de conversión se inicia con un signo de por ciento y termina con un especificador de conversión. En una cadena de control de formato pueden existir muchas especificaciones de conversión.

Error común de programación 9.1

Olvidar encerrar entre comillas una cadena de control de formato.

Práctica sana de programación 9.1

Edite las salidas de manera nítida para su presentación. Esto hace más legible las salidas del programa y reduce errores de usuario.

9.4 Cómo imprimir enteros

Un entero es un número completo, como 776 o -52, que no contiene punto decimal. Los valores enteros se despliegan en uno de varios formatos. En la figura 9.1 se describe cada uno de los especificadores de conversión de enteros.

El programa de la figura 9.2 imprime un entero, utilizando cada uno de los especificadores de conversión de enteros. Note que sólo se imprime el signo menos; los signos más son suprimidos. Más adelante en este capítulo, veremos cómo obligar a que los signos más se impriman. También note que en una computadora con enteros de 2 bytes, el valor **-455** es leído por **%u** y se convierte al valor no signado **65081**.

Especificador de conversión	Descripción
`d`	Despliega un entero decimal signado.
`i`	Muestra un entero decimal signado. (Nota: los especificadores `i` y `d` son diferentes cuando se utilizan con `scanf`).
`o`	Muestra un entero octal no signado.
`u`	Muestra un entero decimal no signado.
`x o bien X`	Muestra un entero hexadecimal no signado. `X` mayúscula hace que sean mostrados los dígitos `0-9` y las letras `A-F`, y `x` minúsculas hacen que se muestren los dígitos `0-9` y `a-f`.
`h o bien l (letra l)`	Se coloca antes de cualquier especificador de conversión de enteros para indicar que se muestra un entero `short`, o bien `long` respectivamente.

Fig. 9.1 Especificadores de conversión de enteros.

```
/* Using the integer conversion specifiers */
#include <stdio.h>

main()
{
   printf("%d\n", 455);
   printf("%i\n", 455);  /* i same as d in printf */
   printf("%d\n", +455);
   printf("%d\n", -455);
   printf("%hd\n", 32000);
   printf("%ld\n", 2000000000);
   printf("%o\n", 455);
   printf("%u\n", 455);
   printf("%u\n", -455);
   printf("%x\n", 455);
   printf("%X\n", 455);
   return 0;
}
```

```
455
455
455
-455
32000
2000000000
707
455
65081
1c7
1C7
```

Fig. 9.2 Cómo utilizar especificadores de conversión de enteros.

Error común de programación 9.2

Imprimir un valor negativo utilizando un especificador de conversión que espera un valor no signado.

9.5 Cómo imprimir números de punto flotante

Un valor de punto flotante contiene un punto decimal, como en `33.5` o `657.983`. Los valores de punto flotante son desplegados en uno de varios formatos. En la figura 9.3 se describen los especificadores de conversión de punto flotante.

Los especificadores de conversión `e` y `E` despliegan valores de punto flotante en *notación exponencial*. La notación exponencial es el equivalente de computadora de la *notación científica* utilizada en matemáticas. Por ejemplo, el valor 150.4582 se representa en notación científica como

$$1.504582 \times 10^2$$

y se representa en notación exponencial como

```
1.504582E+02
```

por la computadora. Esta notación indica que `1.504582` debe ser multiplicado por `10` elevado a la segunda potencia (`E+02`). La `E` significa "exponente".

En forma preestablecida, los valores impresos con los especificadores de conversión `e`, `E`, y `f` son extraídos con 6 dígitos de precisión a la derecha del punto decimal; otras precisiones pueden ser especificadas de forma explícita. El especificador de conversión `f` siempre imprime por lo menos un dígito a la izquierda del punto decimal. Los especificadores de conversión `e`, y `E` imprimen respectivamente la `e` en minúsculas y la `E` en mayúsculas antes del exponente y siempre se imprimen exactamente un dígito a la izquierda del punto decimal.

El especificado de conversión `g` (`G`) imprime en formato `e` (`E`), o en formato `f`, sin ceros después del punto decimal (es decir, `1.234000` se imprime como `1.234`). Los valores se imprimen con `e` (`E`), si después de convertir el valor a notación exponencial, el exponente del valor es menor de `-4`, o es mayor o igual a la precisión especificada (por omisión, en el caso de `g` y de `G`, 6 dígitos significativos). De lo contrario, para imprimir el valor se utilizará el especificador de conversión `f`. Utilizando `g` o `G`, no se imprimen los ceros a la derecha del punto decimal, en la parte fraccionaria de la salida del valor. Para que se extraiga el punto decimal se requiere por lo menos de un dígito decimal.

Especificador de conversión	Descripción
`e o bien E`	Muestra un valor en punto flotante en notación exponencial.
`f`	Muestra valores en punto flotante.
`g o bien G`	Despliega un valor en punto flotante, ya sea en la forma de punto flotante `f`, o en la forma exponencial `e` (o `E`).
`L`	Se coloca antes de cualquier especificador de conversión de punto flotante, para indicar que está mostrado un valor de punto flotante `long double`.

Fig. 9.3 Especificadores de conversión de punto flotante.

Si se utiliza la especificación de conversión %g, los valores 0.0000875, 8750000.0, 8.75, 87.50 y 875 se imprimen como 8.75e-05, 8.75e+06, 8.75, 87.5 y 875. El valor 0.0000875 utiliza notación e porque, al convertirse a notación exponencial, su exponente es menor que -4. El valor 8750000.0 utiliza notación e debido a que su exponente es igual a la precisión por omisión.

La precisión de los especificadores de conversión g y G indican el número máximo de dígitos significativos impresos, incluyendo el dígito a la izquierda del punto decimal. Utilizando la especificación de conversión %g, el valor 1234567.0 se imprime como 1.23457e+06 (recuerde que todos los especificadores de conversión de punto flotante tienen una precisión por omisión de 6). Note que en el resultado aparecen 6 dígitos significativos. La diferencia entre g y G es idéntica a la diferencia entre e y E, cuando el valor se imprime en notación exponencial la g minúscula hace que la salida sea una e, y la G hace que la salida sea una E.

Práctica sana de programación 9.2

Al extraer datos, asegúrese que el usuario está consciente de situaciones en las cuales, debido al formato, los datos pudieran resultar imprecisos (por ejemplo, errores de redondeo debidos a precisiones especificadas).

El programa de la figura 9.4 demuestra cada una de las tres especificaciones de conversión de punto flotante. Advierta que las especificaciones de conversión %E y %g hacen que en la salida el valor se redondee.

```
/* Printing floating-point numbers with
   floating-point conversion specifiers */

#include <stdio.h>

main()
{
   printf("%e\n", 1234567.89);
   printf("%e\n", +1234567.89);
   printf("%e\n", -1234567.89);
   printf("%E\n", 1234567.89);
   printf("%f\n", 1234567.89);
   printf("%g\n", 1234567.89);
   printf("%G\n", 1234567.89);

   return 0;
}
```

```
1.234568e+06
1.234568e+06
-1.234568e+06
1.234568E+06
1234567.890000
1.23457e+06
1.23457E+06
```

Fig. 9.4 Cómo utilizar especificadores de conversión de punto flotante.

9.6 Cómo imprimir cadenas y caracteres

Los especificadores de conversión c y s se utilizan para imprimir respectivamente caracteres individuales y cadenas. El especificador de conversión c requiere de un argumento char. El especificador de conversión s requiere de un apuntador a char como argumento. El especificador de conversión s hace que se impriman caracteres hasta que encuentre un carácter de terminación NULL ('\0'). El programa que se muestra en la figura 9.5 despliega caracteres y cadenas con los especificadores de conversión c y s.

Error común de programación 9.3

*Utilizar %c para imprimir el primer carácter de una cadena. La especificación de conversión %c espera un argumento char. Una cadena es un apuntador a char, es decir, a char *.*

Error común de programación 9.4

Utilizar %s para imprimir un argumento char. La especificación de conversión %s espera un argumento de tipo apuntador a char. En algunos sistemas, esto causará un error fatal en tiempo de ejecución, conocido como violación de acceso.

Error común de programación 9.5

Es un error de sintaxis usar comillas sencillas alrededor de cadenas de caracteres. Las cadenas de caracteres deben estar encerradas entre comillas dobles.

Error común de programación 9.6

Usar comillas dobles alrededor de una constante de carácter. Esto, de hecho, crea una cadena formado por dos caracteres, el segundo de los cuales es el carácter de terminación NULL. Una constante de carácter es un carácter, encerrado entre comillas sencillas.

```
/* Printing strings and characters */
#include <stdio.h>

main()
{
   char character = 'A';
   char string[] = "This is a string";
   char *stringPtr = "This is also a string";

   printf("%c\n", character);
   printf("%s\n", "This is a string");
   printf("%s\n", string);
   printf("%s\n", stringPtr);
   return 0;
}
```

```
A
This is a string
This is a string
This is also a string
```

Fig. 9.5 Cómo utilizar los especificadores de conversión de caracteres y de cadenas.

9.7 Otros especificadores de conversión

Los tres especificadores de conversión restantes son **p**, **n** y % (figura 9.6).

Sugerencia de portabilidad 9.1

El especificador de conversión **p** *despliega una dirección de apuntador en forma de puesta en marcha definida (en muchos sistemas se utiliza notación hexadecimal en preferencia a notación decimal).*

El especificador de conversión **n** almacena el número de caracteres ya extraídos en el enunciado actual **printf**— el argumento correspondiente es un apuntador a una variable entera en la cual se almacena el valor. Mediante una especificación de conversión %**n** nada se imprime. El especificador de conversión % hace que se extraiga un signo de %.

En el programa de la figura 9.7, %**p** imprime el valor de **ptr** y la dirección de **x**; estos valores resultan idénticos porque a **ptr** es asignada a la dirección de **x**. A continuación, %**n** almacena el número de caracteres extraídos por el tercer enunciado **printf** en la variable entera **y**, y es impreso el valor de **y**. El último enunciado **printf** utiliza %% para imprimir el carácter % en una cadena de carácter. Note que cada una de las llamadas **printf** regresa un valor ya sea el número de caracteres extraídos o si ha ocurrido un error de salida, un valor negativo.

Error común de programación 9.7

Intentar imprimir un carácter literal de por ciento, utilizando en la cadena de control de formato % en vez de %%. Cuando en la cadena de control de formato aparece el signo de %, debe estar seguido por un especificador de conversión.

9.8 Cómo imprimir con anchos de campo y precisiones

El tamaño exacto de un campo en el cual se imprimen datos se especifica por el *ancho del campo*. Si el ancho del campo es mayor que los datos que se están imprimiendo, a menudo los datos dentro de dicho campo quedarán justificados a la derecha. Un entero que representa el ancho del campo es insertado en la especificación de conversión, entre el signo de por ciento (%) y el especificador de conversión. El programa en la figura 9.8 imprime dos grupos cada uno de cinco números, justificando a la derecha aquellos números que contienen menos dígitos que el ancho del campo. Note que el ancho de campo es de manera automática aumentado para imprimir valores más anchos que el campo, y que el signo de menos de un valor negativo ocupa en el ancho de campo

Especificador de conversión	Descripción
p	Muestra un valor de apuntador en forma de puesta en marcha definida.
n	Almacena el número de caracteres ya extraídos en el enunciado **printf** actual. Se proporciona un apuntador a un entero como argumento correspondiente. No se muestra nada.
%	Muestra el carácter de por ciento.

Fig. 9.6 Otros especificadores de conversión.


```
/* Using the p, n, and % conversion specifiers */
#include <stdio.h>

main()
{
   int *ptr;
   int x = 12345, y;

   ptr = &x;
   printf("The value of ptr is %p\n", ptr);
   printf("The address of x is %p\n\n", &x);

   printf("Total characters printed on this line is:%n", &y);
   printf(" %d\n\n", y);

   y = printf("This line has 28 characters\n");
   printf("%d characters were printed\n\n", y);

   printf("Printing a %% in a format control string\n");
   return 0;
}
```

```
The value of ptr is 001F2BB4
The address of x is 001F2BB4

Total characters printed on this line is: 41

This line has 28 characters
28 characters were printed

Printing a % in a format control string
```

Fig. 9.7 Cómo utilizar los especificadores de conversión **p**, **n** y %.

una posición de carácter. Los anchos de campo pueden ser utilizados con todos los especificadores de conversión.

Error común de programación 9.8

No proporcionar un ancho de campo lo suficiente extenso para manejar un valor a imprimirse. Esto puede desplazar otros datos imprimiéndose y puede producir salidas confusas. ¡Familiarícese con sus datos!

La función **printf** también da la capacidad de definir la *precisión* con la cual los datos se imprimirán. La precisión tiene significados distintos para diferentes tipos de datos. Cuando se utiliza con especificadores de conversión de enteros, la precisión indica el número mínimo de dígitos a imprimirse. Si el valor impreso contiene menos dígitos que la precisión especificada, al valor impreso se le antepondrán ceros, hasta que el número total de dígitos sea equivalente a la precisión. La precisión por omisión para enteros es 1. Cuando se utiliza con especificadores de conversión de punto flotante **e**, **E** y **f**, la precisión es el número de dígitos que aparecerá después del punto decimal. Cuando se utilice con los especificadores de conversión **g** y **G**, la precisión es

```
/* Printing integers right-justified */
#include <stdio.h>

main()
{
   printf("%4d\n", 1);
   printf("%4d\n", 12);
   printf("%4d\n", 123);
   printf("%4d\n", 1234);
   printf("%4d\n\n", 12345);

   printf("%4d\n", -1);
   printf("%4d\n", -12);
   printf("%4d\n", -123);
   printf("%4d\n", -1234);
   printf("%4d\n", -12345);

   return 0;
}
```

```
   1
  12
 123
1234
12345

  -1
 -12
-123
-1234
-12345
```

Fig. 9.8 Justificación de enteros a la derecha en un campo.

el número máximo de dígitos significativos a imprimirse. Cuando se utiliza con el especificador de conversión **s**, la precisión es el número máximo de caracteres a escribirse a partir de una cadena. Para utilizar la precisión, coloque un punto decimal (.) seguido por un entero que representa la precisión, entre el signo de por ciento y el especificador de conversión. El programa de la figura 9.9 demuestra el uso de la precisión en cadenas de control de formato. Note que cuando un valor en punto flotante se imprime con una precisión menor en el valor que el número original de decimales, el valor quedará redondeado.

Se pueden combinar el ancho de campo y la precisión, colocando el ancho de campo seguido por el punto decimal, seguido por la precisión, entre el signo de por ciento y el especificador de conversión, como en el enunciado

```
printf("%9.3f", 123.456789);
```

que despliega **123.4567** con 3 dígitos a la derecha del punto decimal, y queda justificado a la derecha, en un campo de 9 dígitos.


```
/* Using precision while printing integers,
   floating-point numbers, and strings */
#include <stdio.h>

main()
{
   int i = 873;
   float f = 123.94536;
   char s[] = "Happy Birthday";

   printf("Using precision for integers\n");
   printf("\t%.4d\n\t%.9d\n\n", i, i);
   printf("Using precision for floating-point numbers\n");
   printf("\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f);
   printf("Using precision for strings\n");
   printf("\t%.11s\n", s);
   return 0;
}
```

```
Using precision for integers
        0873
        000000873

Using precision for floating-point numbers
        123.945
        1.239e+02
        124

Using precision for strings
        Happy Birth
```

Fig. 9.9 Cómo usar precisiones para mostrar información de varios tipos.

Utilizando expresiones enteras en la lista de argumentos, a continuación de la cadena de control de formato, es posible especificar el ancho de campo y la precisión. Para utilizar esta característica, inserte un * (asterisco) en el lugar del ancho de campo o de la precisión (o de ambos). El argumento coincidente en la lista de argumentos se evalúa y se utiliza en lugar del asterisco. El valor del argumento puede ser negativo para el ancho de campo, pero debe ser positivo para la precisión. Un valor negativo para el ancho de campo hace que la salida se justifique a la izquierda en el campo, tal y como se describe en la sección siguiente. El enunciado

```
printf("%*.*f", 7, 2, 98.736);
```

utiliza **7** para el ancho de campo, **2** para la precisión y extrae el valor **98.74** justificado a la derecha.

9.9 Uso de banderas en la cadena de control de formato de printf

La función **printf** tiene también *banderas* para complementar sus capacidades de formato de salida. Están disponibles para el usuario cinco banderas, para uso en cadenas de control de formato (figura 9.10).

Bandera	Descripción
- (signo de menos)	Justificación a la izquierda de la salida, dentro del campo especificado.
+ (signo más)	Despliega un signo más, antes de valores positivos y un signo menos, antes de valores negativos.
espacio	Imprime un espacio antes de un valor positivo que no se imprima con la bandera +.
#	Antecede un 0 al valor extraído, cuando se utiliza con el especificador de conversión octal o.
	Antecede 0x o 0X al valor de salida, cuando se utiliza con los especificadores de conversión hexadecimales x o X.
	Obliga a un punto decimal para un número de punto decimal impreso con e, E, f, g o G, que no contenga una parte fraccionaria. (Por lo regular sólo se imprime el punto decimal si le sigue algún dígito). En el caso de los especificadores g y G, no se eliminan los ceros a la derecha.
0 (cero)	Rellena un campo con ceros a la izquierda.

Fig. 9.10 Banderas de cadenas de control de formato.

Para utilizar una bandera en una cadena de control de formato, colóquela de inmediato a la derecha del signo de por ciento. En una especificación de conversión se pueden combinar varias banderas.

El programa de la figura 9.11 demuestra la justificación a la derecha y a la izquierda de una cadena, de un entero, de un carácter y de un número de punto flotante.

El programa de la figura 9.12 imprime un número positivo y uno negativo, cada uno de ellos con y sin la bandera +. Note que en ambos casos el signo de menos queda desplegado, pero el signo de más sólo es desplegado cuando se utiliza la bandera más.

El programa de la figura 9.13 antepone un espacio al número positivo con la bandera de espacio. Esto resulta útil para alinear números positivos y negativos con el mismo número de dígitos.

```
/* Right justifying and left justifying values */
#include <stdio.h>

main()
{
   printf("%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23);
   printf("%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23);
   return 0;
}
```

```
     hello         7         a  1.230000

hello     7         a         1.230000
```

Fig. 9.11 Cómo justificar a la izquierda cadenas en un campo.


```
/* Printing numbers with and without the + flag */
#include <stdio.h>

main()
{
   printf("%d\n%d\n", 786, -786);
   printf("%+d\n%+d\n", 786, -786);
   return 0;
}
```

```
786
-786
+786
-786
```

Fig. 9.12 Cómo imprimir números positivos y negativos con y sin la bandera +.

```
/* Printing a space before signed values
   not preceded by + or - */
#include <stdio.h>

main()
{
   printf("% d\n% d\n", 547, -547);
   return 0;
}
```

```
 547
-547
```

Fig. 9.13 Cómo utilizar la bandera espacio.

El programa de la figura 9.14 utiliza la bandera # para anteceder 0 al valor octal, 0x y 0X a valores hexadecimales, y para obligar a el punto decimal en un valor impreso utilizando a g.

El programa en la figura 9.15 combina la bandera + y la bandera 0 (cero) para imprimir 452 en un campo de 9 espacios, con un signo de + y ceros a la izquierda, y a continuación vuelve a imprimir 452 utilizando sólo la bandera 0 y el campo de 9 espacios.

9.10 Cómo imprimir literales y secuencias de escape

La mayor parte de los caracteres literales a imprimirse en un enunciado **printf** pueden ser incluidos en la cadena de control de formato. Sin embargo, existen varios caracteres "problema" tal y como los signos de comillas ("), que delimitan la cadena de control de formato misma. Varios caracteres de control, como son la nueva línea y el tabulador, deben ser representados por *secuencias de escape*. Una secuencia de escape es representada por una diagonal invertida (\) seguida por un carácter de escape particular. La tabla en la figura 9.16 enlista todas las secuencias de escape y las acciones que éstas causan.

```
/* Using the # flag with conversion specifiers
   o, x, X, and any floating-point specifier */
#include <stdio.h>

main()
{
   int c = 1427;
   float p = 1427.0;

   printf("%#o\n", c);
   printf("%#x\n", c);
   printf("%#X\n", c);
   printf("\n%g\n", p);
   printf("%#g\n", p);
   return 0;
}
```

```
02623
0x593
0X593

1427
1427.00
```

Fig. 9.14 Cómo utilizar la bandera # .

```
/* Printing with the 0(zero) flag fills in leading zeros */
#include <stdio.h>

main()
{
   printf("%+09d", 452);
   printf("%09d", 452);
   return 0;
}
```

```
+00000452
000000452
```

Fig. 9.15 Cómo utilizar la bandera 0 (cero).

Error común de programación 9.9

Intentar imprimir como datos literales en un enunciado `printf` *una comilla, dobles comillas, signo de interrogación o un carácter de diagonal invertida, sin anteceder dichos caracteres por una diagonal invertida para formar una secuencia de escape correcta.*

Secuencia de escape	Descripción
\'	Salida del carácter de una sola comilla (').
\"	Salida del carácter de dobles comillas (").
\?	Salida del signo de interrogación (?).
\\	Salida del carácter de diagonal invertida (\).
\a	Genera una alerta visual o audible (campana).
\b	Mueve el cursor hacia atrás una posición en la línea actual.
\f	Mueve el cursor al inicio de la siguiente página lógica.
\n	Mueve el cursor al inicio de la línea siguiente.
\r	Mueve el cursor al principio de la línea actual.
\t	Mueve el cursor a la siguiente posición de tabulador horizontal.
\v	Mueve el cursor a la siguiente posición de tabulador vertical.

Fig. 9.16 Secuencias de escape.

9.11 Formato de entrada utilizando scanf

Se consigue entrada con formato preciso utilizando **`scanf`**. Cada enunciado **`scanf`** contiene una cadena de control de formato que describe el formato de los datos que se introducen. La cadena de control está formada de especificaciones de conversión y de caracteres literales. La función **`scanf`** tiene las siguientes capacidades de formato de entrada:

1. Entrada de todo tipo de datos.
2. Entrada de caracteres específicos desde un flujo de entrada.
3. Omitir caracteres específicos del flujo de entrada.

La función **`scanf`** se escribe en la forma siguiente:

`scanf` (*format-control-string, other-arguments*) ;

Format-control-string describe los formatos de la entrada, y *other-arguments* son apuntadores a variables, en los cuales se almacena la entrada.

Práctica sana de programación 9.3

Al introducir datos, solicite al usuario un elemento o pocos elementos de datos a la vez. Evite solicitar al usuario la introducción de muchos elementos de datos, en respuesta a una solicitud.

En la figura 9.17 se resumen los especificadores de conversión utilizados para introducir todos los tipos de datos. El resto de esta sección presenta programas que demuestran la lectura de datos utilizando los varios especificadores de conversión de **`scanf`**.

El programa en la figura 9.18 lee enteros con varios especificadores de conversión de enteros, y despliega los enteros como números decimales. Note que **`%i`** es capaz de introducir enteros decimales, octales y hexadecimales.

Especificador de conversión	Descripción
Enteros	
`d`	Lee un entero decimal, opcionalmente signado. El argumento correspondiente es un apuntador a un entero.
`i`	Lee un entero decimal, octal o hexadecimal, opcionalmente signado. El argumento correspondiente es un apuntador a un entero.
`o`	Lee un entero octal. El argumento correspondiente es un apuntador a un entero no signado.
`u`	Lee un entero decimal no signado. El argumento correspondiente es un apuntador a un entero no signado.
`x` o `X`	Lee un entero hexadecimal. El argumento correspondiente es un apuntador a un entero no signado.
`h` o `l`	Se coloca antes de cualquiera de los especificadores de conversión de enteros, para indicar que un entero **`short`** o **`long`** será introducido.
Números de punto flotante	
`e, E, f, g,` o `G`	Lee un valor en punto flotante. El argumento correspondiente es un apuntador a una variable de punto flotante.
`l` o `L`	Se coloca delante de cualquier especificador de conversión de punto flotante para indicar que un valor **`double`** o **`long double`** será introducido.
Caracteres y cadenas.	
`c`	Lee un carácter. El argumento correspondiente es un apuntador a **`char`** y **`NULL`** (`'\0'`) no se agrega.
`s`	Lee una cadena. El argumento correspondiente es un apuntador a un arreglo del tipo **`char`**, que es lo suficiente extenso para contener la cadena y un carácter de terminación **`NULL`** (`'\0'`).
Conjunto de rastreo *[scan characters]*	Rastrea una cadena buscando un conjunto de caracteres almacenados en un arreglo.
Misceláneos	
`p`	Lee una dirección de apuntador producido de la misma forma que cuando una dirección es extraída con `%p` en un enunciado **`printf`**.
`n`	Almacena el número de caracteres introducidos hasta el momento en este **`scanf`**. El argumento correspondiente es un apuntador a entero.
`%`	Saltarse un signo de por ciento (%) en la entrada.

Fig. 9.17 Especificadores de conversión para **`scanf`**.

Al introducir números de punto flotante, cualquiera de los especificadores de conversión de punto flotante **`e`**, **`E`**, **`f`**, **`g`**, o **`G`** pueden ser utilizados. El programa de la figura 9.19 demuestra la lectura de tres números de punto flotante, con cada uno de los tres tipos de los especificadores flotantes de conversión y muestra todos los tres números mediante el especificador de conversión **`f`**. Note que la salida del programa confirma el hecho que los valores del punto flotante no son precisos —este hecho queda resaltado en el segundo valor impreso.


```
/* Reading integers */
#include <stdio.h>

main()
{
   int a, b, c, d, e, f, g;

   printf("Enter seven integers: ");
   scanf("%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g);
   printf("The input displayed as decimal integers is:\n");
   printf("%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g);
   return 0;
}
```

```
Enter seven integers: -70 -70 070 0x70 70 70 70
The input displayed as decimal integers is:
-70 -70 56 112 56 70 112
```

Fig. 9.18 Cómo leer entradas con especificadores de conversión a enteros.

```
/* Reading floating-point numbers */
#include <stdio.h>

main()
{
   float a, b, c;

   printf("Enter three floating-point numbers: \n");
   scanf("%e%f%g", &a, &b, &c);
   printf("Here are the numbers entered in plain\n");
   printf("floating-point notation:\n");
   printf("%f %f %f\n", a, b, c);
   return 0;
}
```

```
Enter three floating-point numbers:
1.27987 1.27987e+03 3.38476e-06
Here are the numbers entered in plain
floting-point notation:
1.279870
1279.869995
0.000003
```

Fig. 9.19 Cómo leer entradas con especificadores de conversión de punto flotante.

Los caracteres y cadenas se introducen utilizando los especificadores de conversión **`c`** y **`s`**, respectivamente. El programa de la figura 9.20 solicita al usuario que introduzca una cadena. El programa introduce el primer carácter de la cadena utilizando `%c` y lo almacena en la variable de

carácter **x**, y a continuación introduce el resto de la cadena, utilizando **%s** y lo almacena en el arreglo de caracteres **y**.

Se puede introducir una secuencia de caracteres utilizando un *conjunto de rastreo*. Un conjunto de rastreo es un conjunto de caracteres encerrados en corchetes [], y precedidos por un signo por ciento en la cadena de control de formato. Un conjunto de rastreo rastrea los caracteres en el flujo de entrada, buscando sólo aquellos caracteres que coincidan con los caracteres contenidos en el conjunto de rastreo. Cada vez que se encuentre un carácter, se almacena en el argumento correspondiente del conjunto de caracteres que es un apuntador a un arreglo de caracteres. El conjunto de rastreo deja de introducir caracteres cuando encuentra un carácter que no esté contenido en el conjunto de rastreo. Si el primer carácter en el flujo de entrada no coincide con un carácter en el conjunto de rastreo, sólo se almacenará en el arreglo el carácter nulo. El programa en la figura 9.21 utiliza un conjunto de rastreo **[aeiou]** para rastrear el flujo de entrada buscando vocales. Observe que se leerán las primeras siete letras de la entrada. La octava letra (**h**) no forma parte del conjunto de rastreo y, por lo tanto, el rastreo terminará.

El conjunto de rastreo también puede ser utilizado para rastrear buscando caracteres que no estén contenidos en el mismo, utilizando un *conjunto de rastreo invertido*. Para crear un conjunto de rastreo invertido, coloque un *acento circunflejo* (^) en los corchetes antes de los caracteres de rastreo. Esto hará que sean almacenados los caracteres que no aparecen en el conjunto de rastreo. Cuando se detecte un carácter contenido en el conjunto de rastreo invertido, se terminará la entrada. El programa en la figura 9.22 utiliza el conjunto de rastreo invertido **[^aeiou]** para buscar consonantes o más apropiadamente para buscar "no vocales".

En la especificación de conversión **scanf** se puede utilizar un ancho de campo para leer un número específico de caracteres a partir del flujo de entrada. El programa de la figura 9.23 introduce una serie de dígitos consecutivos como un entero de dos dígitos y un entero que consiste de los dígitos restantes en el flujo de entrada.

```
/* Reading characters and strings */
#include <stdio.h>

main()
{
   char x, y[9];

   printf("Enter a string: ");
   scanf("%c%s", &x, y);

   printf("The input was:\n");
   printf("the character \"%c\" ", x);
   printf("and the string \"%s\"\n", y);
   return 0;
}
```

```
Enter a string: Sunday
The input was:
the character "S" and the string "unday"
```

Fig. 9.20 Cómo introducir caracteres y cadenas.


```
/* Using a scan set */
#include <stdio.h>

main()
{
   char z[9];

   printf("Enter string: ");
   scanf("%[aeiou]", z);
   printf("The input was \"%s\"\n", z);
   return 0;
}
```

```
Enter String: ooeeooahah
The input was "ooeeooa"
```

Fig. 9.21 Cómo usar un conjunto de rastreo.

```
/* Using an inverted scan set */
#include <stdio.h>

main()
{
   char z[9];

   printf("Enter a string: ");
   scanf("%[^aeiou]", z);
   printf("The input was \"%s\"\n", z);
   return 0;
}
```

```
Enter a string: String
The input was "Str"
```

Fig. 9.22 Cómo utilizar un conjunto de rastreo invertido.

A menudo es necesario omitir ciertos caracteres del flujo de entrada. Por ejemplo, una fecha podía ser introducida como

```
7-9-91
```

Es necesario almacenar cada número de la fecha, pero los guiones que separan los números pueden ser descartados. A fin de eliminar caracteres innecesarios, inclúyalos en la cadena de control de formato de **scanf** (caracteres de espacio en blanco —como es el espacio en blanco, la nueva línea y el tabulador— pasar por alto todos los espacios en blanco a la izquierda). Por ejemplo, para pasar por alto los guiones en la entrada, utilice el enunciado

```
scanf("%d-%d-%d", &month, &day, &year);
```

```
/* inputting data with a field width */
#include <stdio.h>

main()
{
   int x, y;

   printf("Enter a six digit integer: ");
   scanf("%2d%d", &x, &y);
   printf("The integers input were %d and %d\n", x, y);
   return 0;
}
```

```
Enter a six digit integer: 123456
The integers input were 12 and 3456
```

Fig. 9.23 Cómo introducir datos con un ancho de campo.

Aunque este **scanf** sí elimina los guiones de la entrada anterior, es posible que la fecha hubiera sido escrita como

```
7/9/91
```

En este caso el **scanf** precedente no eliminaría caracteres innecesarios. Por esta razón, **scanf** proporciona un *carácter de supresión de asignación* *. El carácter de supresión de asignación, le permite a **scanf** leer cualquier tipo de datos a partir de la entrada y descartarlos sin asignarlos a una variable. El programa de la figura 9.24 utiliza el carácter de supresión de asignación en la especificación de conversión **%c**, para indicar que un carácter que aparece en el flujo de entrada deberá ser leído y descartado. Sólo se almacenará mes, día y año. Los valores de las variables se imprimen para demostrar que de hecho han sido introducidos en forma correcta. Note que ninguna variable en la lista de argumentos corresponde a las especificaciones de conversión que utiliza el carácter de supresión de asignación, porque con estas especificaciones de conversión no se efectúa ninguna asignación.

Resumen

- Toda entrada y salida se maneja a través de flujos —secuencias de caracteres organizadas en línea. Cada línea consiste de cero o más caracteres y terminan con el carácter de nueva línea.
- Por lo regular, el flujo estándar de entrada se conecta al teclado, y el flujo estándar de salida está conectado a la pantalla de la computadora.
- Los sistemas operativos a menudo permiten que los flujos de entrada y salida estándar, sean redirigidos a otros dispositivos.
- La cadena de control de formato **printf** describe el formato o formatos en los cuales aparecerán los valores de salida. La cadena de control de formato está formada de los especificadores de conversión, las banderas, los anchos de campo, las precisiones y los caracteres literales.


```
/* Reading and discarding characters from the input stream */
#include <stdio.h>

main()
{
   int month1, day1, year1, month2, day2, year2;

   printf("Enter a date in the form mm-dd-yy: ");
   scanf("%d%*c%d%*c%d", &month1, &day1, &year1);
   printf("month = %d  day = %d  year = %d\n\n",
      month1, day1, year1);
   printf("Enter a date in the form mm/dd/yy: ");
   scanf("%d%*c%d%*c%d", &month2, &day2, &year2);
   printf("month = %d  day = %d  year = %d\n",
      month2, day2, year2);
   return 0;
}
```

```
Enter a date in the form mm-dd-yy: 11-18-71
month = 11  day = 18  year = 71

Enter a date in the form mm/dd/yy: 11/18/71
month = 11  day = 18  year = 71
```

Fig. 9.24 Cómo leer y descartar caracteres del flujo de entrada.

- Los enteros se imprimen con los especificadores de conversión siguientes : **d** o **i** para enteros opcionalmente signados, o para enteros no signados en forma octal, **u** para enteros no signados en forma decimal, y **x** o bien **X** para enteros no signados en forma hexadecimal. Para indicar un entero **short** o **long** respectivamente a los especificadores de conversión ya descritos, se antecede el modificador **h** o **l**.
- Se imprimen los valores en punto flotante utilizando los especificadores de conversión siguientes: **e** o **E** en el caso de notación exponencial, **f** para la notación de punto flotante normal y **g** o **G** para ya sea **e** o (**E**) o para la notación **f**. Cuando se indica el especificador de conversión **g** o (**G**), si el exponente del valor es menor de **-4** o mayor o igual que la precisión con la cual se imprime el valor, se utiliza el especificador de conversión **e** o (**E**).
- La precisión para los especificadores de conversión **g** y **G** indica el número máximo de dígitos significativos que se imprimirán.
- El especificador de conversión **c** imprime un carácter.
- El especificador de conversión **s** imprime una cadena de caracteres terminadas por un carácter nulo.
- El especificador de conversión **p** despliega una dirección de apuntador en forma de puesta en marcha definida (en muchos sistemas se utiliza la notación hexadecimal).
- El especificador de conversión **n** almacena el número de caracteres ya salido en el enunciado actual **printf**. El argumento correspondiente es un apuntador a un entero.

- El especificador de conversión %% hace que salga un % literal.
- Si el ancho de campo es mayor que el objeto que se está imprimiendo, el objeto quedará justificado a la derecha dentro de dicho campo.
- Los anchos de campo pueden ser utilizados por todos los especificadores de conversión.
- La precisión utilizada con los especificadores de conversión enteros indican el número mínimo de dígitos a imprimirse. Si el valor contiene menos dígitos que la precisión especificada, se antecederán ceros al valor impreso, hasta que el número de dígitos sea equivalente a la precisión.
- La precisión utilizada con especificadores de conversión de punto flotante **e**, **E**, y **f** indica el número de dígitos que aparecerán después del punto decimal.
- La precisión utilizada con especificadores de conversión de punto flotante **g** y **G** indica el número de dígitos significativos que aparecerán.
- La precisión utilizada con el especificador de conversión s indica el número de caracteres a imprimirse.
- El ancho de campo y la precisión pueden ser combinados, colocando el ancho de campo seguido por un punto decimal y a su vez seguido por la precisión, entre el % y el especificador de conversión.
- Es posible especificar el ancho de campo y la precisión mediante expresiones enteras en la lista de argumentos, a continuación de la cadena de control de formato. Para utilizar esta característica, inserte un * (asterisco) en lugar del ancho de campo o de la precisión. El argumento coincidente en la lista de argumentos se evalúa y se utilizará en lugar del asterisco. El valor del argumento puede ser negativo para el ancho de campo, pero para la precisión deberá ser positivo.
- La bandera signo de - justifica a la izquierda su argumento dentro de un campo.
- La bandera + imprime un signo más para valores positivos y un signo menos para valores negativos.
- La bandera de espacio imprime un espacio antes de un valor positivo, que no esté mostrado con la bandera de +.
- La bandera de signo de # antecede un 0 a los valores octal, **0x** o **0X** a los valores hexadecimales, y obliga a que se imprima el punto decimal en el caso de valores de punto flotante impresos con **e**, **E**, **f**, **g**, o **G** (normalmente el punto decimal se desplegará sólo si el valor contiene una parte fraccionaria).
- La bandera 0 imprime ceros a la izquierda para un valor que no ocupa todo su ancho de campo.
- Se consigue un formato preciso de entrada utilizando la función de biblioteca **scanf**.
- Los enteros se introducen mediante los especificadores de conversión **d** e **i** para enteros opcionalmente signados, y **o**, **u**, **x**, o **X** para enteros no signados. Para introducir un entero **short** o bien **long** respectivamente, se colocan los modificadores **h** y **l** antes de un especificador de conversión entero.
- Los valores de punto flotante son introducidos con los especificadores de conversión **e**, **E**, **f**, **g**, o **G**. Se colocan los modificadores **l** y **L** antes de cualquiera de los especificadores de conversión de punto flotante, para indicar que el valor introducido es un valor **double**, o bien **long double**, respectivamente.
- Los caracteres se introducen mediante el especificador de conversión **c**.
- Las cadenas se introducen mediante el especificador de conversión **s**.

- Un conjunto de rastreo, rastrea los caracteres en la entrada buscando sólo aquellos caracteres que coincidan con los caracteres que contiene el conjunto de rastreo. Cuando se encuentra un carácter, se almacena un arreglo de caracteres. El conjunto de rastreo deja de introducir caracteres cuando es encontrado un carácter no contenido en el conjunto de rastreo.
- Para crear un conjunto de rastreo invertido, coloque un acento circunflejo (^) en los corchetes, antes de los caracteres de rastreo. Esto hace que los caracteres que no aparecen en el conjunto de rastreo sean almacenados hasta que se encuentre un carácter contenido en el conjunto de rastreo invertido.
- Los valores de dirección se introducen utilizando el especificador de conversión **p**.
- El especificador de conversión **n** almacena el número de caracteres ya introducido en el **scanf** actual. El argumento correspondiente es un apuntador a **int**.
- La especificación de conversión %% hace coincidir en la entrada un carácter único de %.
- El carácter de supresión de asignación se utiliza para leer datos del flujo de entrada y descartar dichos datos.
- Un ancho de campo se puede utilizar en un **scanf** para leer un número específico de caracteres a partir de flujo de entrada.

Terminología

bandera #
especificador de conversión %
* en ancho de campo
* en precisión
bandera + (signo de más)
bandera - (signo de menos)
bandera 0 (cero)
<stdio.h>
secuencia de escape \"
secuencia de escape \'
secuencia de escape \?
secuencia de escape \\
secuencia de escape \a
secuencia de escape \b
secuencia de escape \f
secuencia de escape \n
secuencia de escape \r
secuencia de escape \t
secuencia de escape \v
alineación
carácter de supresión de asignación (*)
inserción de espacio en blanco
especificador de conversión c
acento circunflejo (^)
especificación de conversión
especificadores de conversión
especificador de conversión d
especificador de conversión e o E
secuencia de escape
formato exponencial de punto flotante
especificador de conversión f
ancho de campo
bandera
punto flotante
cadena de control de formato
especificador de conversión g o G
especificador de conversión h
formato hexadecimal
especificador de conversión i
especificadores de conversión enteros
conjunto de rastreo invertido
especificador de conversión L
especificador de conversión l
justificación a la izquierda
caracteres literales
entero long
especificador de conversión n
especificador de conversión o
formato octal
especificador de conversión p
precisión
printf
inserción de carácter de impresión
redireccionamiento de un flujo
justificación a la derecha
redondeo
especificador de conversión s
conjunto de rastreo

scanf
notación científica
entero **short**
formato entero signado
bandera de espacio
flujo estándar de error
flujo de entrada estándar
flujo de salida estándar
flujo
especificador de conversión **u**
formato de entero no signado
espacio en blanco
especificador de conversión **x** (o **X**)

Errores comunes de programación

9.1 Olvidar encerrar entre comillas una cadena de control de formato.

9.2 Imprimir un valor negativo utilizando un especificador de conversión que espera un valor no signado.

9.3 Utilizar **%c** para imprimir el primer carácter de una cadena. La especificación de conversión **%c** espera un argumento **char**. Una cadena es un apuntador a **char**, es decir, a **char ***.

9.4 Utilizar **%s** para imprimir un argumento **char**. La especificación de conversión **%s** espera un argumento de tipo apuntador a **char**. En algunos sistemas, esto causará un error fatal en tiempo de ejecución, conocido como violación de acceso.

9.5 Es un error de sintaxis usar comillas sencillas alrededor de cadenas de caracteres. Las cadenas de caracteres deben estar encerradas entre comillas dobles.

9.6 Usar comillas dobles alrededor de una constante de carácter. Esto, de hecho, crea una cadena formada por dos caracteres, el segundo de los cuales es el carácter de terminación **NULL**. Una constante de carácter es un carácter, encerrado entre comillas sencillas.

9.7 Intentar imprimir un carácter literal de por ciento, utilizando en la cadena de control de formato % en vez de %%. Cuando en la cadena de control de formato aparece el signo de %, debe ser seguida por un especificador de conversión.

9.8 No proporcionar un ancho de campo lo suficiente extenso para manejar un valor a imprimirse. Esto puede desplazar otros datos imprimiéndose y puede producir salidas confusas. ¡Familiarícese con sus datos!

9.9 Intentar imprimir como datos literales en un enunciado **printf** una comilla, dobles comillas, signo de interrogación o un carácter de diagonal invertida, sin anteceder dichos caracteres por una diagonal invertida para formar una secuencia de escape correcta.

Prácticas sanas de programación

9.1 Edite las salidas de manera nítida para su presentación. Esto hace más legible las salidas del programa y reduce errores de usuario.

9.2 Al extraer datos, asegúrese que el usuario está consciente de situaciones en las cuales, debido al formato, los datos pudieran resultar imprecisos (por ejemplo, errores de redondeo debidos a precisiones especificadas).

9.3 Al introducir datos, solicite al usuario un elemento o pocos elementos de datos a la vez. Evite solicitar al usuario la introducción de muchos elementos de datos, en respuesta a una sola solicitud.

Sugerencias de portabilidad

9.1 El especificador de conversión **p** despliega una dirección de apuntador en forma de puesta en marcha definida (en muchos sistemas se utiliza notación hexadecimal en preferencia a notación decimal).

Ejercicios de autoevaluación

9.1 Llene los espacios de cada uno de los siguientes:

a) Todas las entradas y salidas se manejan en forma de _____.
b) El flujo de _____ se conecta por lo regular al teclado
c) El flujo de _____ se conecta por lo regular a la pantalla de la computadora.
d) Un formato preciso de salida se consigue con la función _____.
e) La cadena de control de formato puede contener _____, _____, _____, _____, y _____.
f) El especificador de conversión _____, o bien _____ puede ser utilizado para la salida de un entero decimal signado.
g) Los especificadores de conversión _____, _____, y _____ se utilizan para desplegar enteros no signados en forma octal, decimal y hexadecimal, respectivamente.
h) Se colocan los modificadores _____, y _____ antes de los especificadores de conversión enteros, a fin de indicar que se despliegan valores enteros **short**, o bien **long**.
i) El especificador de conversión _____ se utiliza para desplegar un valor en punto flotante en notación exponencial.
j) El modificador _____ se coloca antes de cualquier especificador de conversión de punto flotante para indicar que se va a delegar un valor **long double**.
k) Si no se ha especificado precisión, los especificadores de conversión **e**, **E** y **f** se despliegan con _____ dígitos de precisión a la derecha del punto decimal.
l) Para imprimir cadenas y caracteres respectivamente se utilizan los especificadores de conversión _____, y _____.
m) Todas las cadenas terminan con el carácter _____.
n) El ancho de campo y la precisión en una especificación de conversión **printf** pueden ser controladas con expresiones enteras, substituyendo un _____ en lugar del ancho de campo o de la precisión, y colocando la expresión entera en el argumento correspondiente de la lista de argumento.
o) La bandera _____ hace que en un campo la salida quede justificada a la izquierda.
p) La bandera _____ hace que se desplieguen valores, ya sea con un signo de más o con un signo de menos.
q) Se consigue un formato de entrada preciso con la función _____.
r) Se utiliza un _____ para rastrear una cadena buscando caracteres específicos y almacenar dichos caracteres en un arreglo.
s) El especificador de conversión _____ puede ser utilizado para introducir enteros octales, decimales y hexadecimales opcionalmente signados.
t) El especificador de conversión _____ puede ser utilizado para introducir un valor **double**.
u) El _____ se utiliza para leer datos del flujo de entrada y descartarlos sin asignarlos a una variable.
v) Se puede utilizar un _____ en una especificación de conversión **scanf** para indicar que un número específico de caracteres o de dígitos debe ser leído a partir del flujo de entrada.

9.2 Encuentre el error en cada uno de los siguientes y explique cómo puede ser corregido.

a) El siguiente enunciado deberá imprimir el carácter **'c'**

```
printf(" %s\n", 'c');
```

b) El siguiente enunciado debería imprimir **9.375%**

```
printf("%.3f%", 9.375);
```

c) El siguiente enunciado debería imprimir el primer carácter de la cadena **"Monday"**

```
printf("%c\n", "Monday");
```

d) `printf(""A string in quotes"");`
e) `printf(%d%e, 12, 20);`
f) `printf("%c", "x");`
g) `printf("%s\n", 'Richard');`

9.3 Escriba un enunciado para cada uno de los siguientes:

a) Imprima `1234`, justificados a la derecha, en un campo de `10` dígitos.
b) Imprima `123.456789`, en notación exponencial, con un signo de (+ o de -) y `3` dígitos de precisión.
c) Lea un valor `double` a la variable `number`.
d) Imprima `100` en forma octal, precedida por `0`.
e) Lea una cadena del arreglo de caracteres `string`.
f) Lea caracteres al arreglo `n`, hasta que se encuentre un carácter no dígito.
g) Utilice las variables enteras `x` e `y`, para especificar un ancho de campo y una precisión utilizada para desplegar el valor `double 87.4573`.
h) Lea un valor de la forma `3.5%`. Almacene el porcentaje en la variable `float percent`, y elimine el signo de % del flujo de entrada. No utilice el carácter de supresión de asignación.
i) Imprima `3.333333` como un valor `long double` con un signo ,(+ o -) en un campo de `20` caracteres con una precisión de `3`.

Respuestas a los ejercicios de autoevaluación

9.1 a) Flujos. b) Entrada estándar. c) Salida estándar. d) `printf`. e) Especificadores de conversión, bandera, anchos de campo, precisiones y caracteres literales. f) `d`, `i`. g) `o`, `u`, `x` (o `X`). h) `h`, `l`. i) `e` (o `E`). j) `L`. k) 6. l) `s`, `c`. m) `NULL('\0')`. n) asterisco (`*`). o) - (menos). p) + (más). q) `scanf`. r) Conjunto de rastreo. s) `i`. t) `1e`, `1E`, `1f`, `1g`, o bien `1G`. u) Carácter de supresión de asignación (`*`). v) Ancho de campo.

9.2

a) Error: el especificador de conversión `s` espera un argumento del tipo apuntador a `char`.
Corrección: para imprimir el carácter `'c'`, utilice la especificación de conversión %c, o bien cambie `'c'` a `"c"`.
b) Intentar imprimir el carácter literal % sin utilizar la especificador de conversión %%.
Corrección: Utilice %% para imprimir un carácter literal %.
c) Error: el especificador de conversión `c` espera un argumento del tipo `char`.
Corrección: Para imprimir el primer carácter de `"Monday"` utilice el especificador de conversión `%ls`.
d) Error: tratar de imprimir el carácter literal ", sin utilizar la secuencia de escape \".
Corrección: Remplace cada comilla en el conjunto interior de comillas, con \".
e) Error: la cadena de control de formato no está encerrada entre comillas dobles.
Corrección: Encierre `%d%d` entre comillas dobles.
f) El carácter `x` está encerrado entre comillas dobles.
Corrección: los constantes de caracteres a ser impresos utilizando %c deben estar encerrados entre comillas sencillas.
g) Error: la cadena a imprimirse está encerrada entre comillas sencillas.
Corrección: utilice comillas dobles en vez de sencillas para representar a una cadena.

9.3

a) `printf("%10d\n", 1234);`
b) `printf("%+.3e\n", 123.456789);`
c) `scanf("%lf", &number);`
d) `printf("%#o\n", 100);`
e) `scanf("%s", string);`
f) `scanf("%[^0123456789]", n);`
g) `printf("%*.*f\n", x, y, 87.4573);`
h) `scanf("%f%%", &percent);`
i) `printf("%+20.3Lf\n", 3.333333);`

Ejercicios

9.4 Escriba un enunciado `printf` o bien `scanf` para cada uno de los siguientes:

a) Imprima el entero no signado `40000` justificado a la izquierda, en un campo de `15` dígitos, con `8` dígitos.
b) Lea un valor hexadecimal a la variable `hex`.
c) Imprima `200` con y sin un signo.
d) Imprima `100` en forma hexadecimal precedido por `0x`.
e) Lea caracteres al arreglo `s`, hasta que se encuentre con la letra `p`.
f) Imprima `1.234` en un campo de `9` dígitos, con ceros a la izquierda.
g) Lea una hora de la forma `hh:mm:ss` almacenando las partes de la hora en las variables enteras `hour`, `minute` y `second`. Omita los dobles puntos (:) del flujo de entrada. Utilice el carácter de supresión de asignación.
h) Lea una cadena de la forma `"characters"` de la entrada estándar. Almacene la cadena en el arreglo de carácter `s`. Elimine las comillas del flujo de entrada.
i) Lea una hora de la forma `hh:mm:ss` almacenando las partes de la hora en las variables enteras `hour`, `minute` y `second`. Omita los dobles puntos (:) en el flujo de entrada. No utilice el carácter de supresión de asignación.

9.5 Muestre lo que se imprime mediante cada uno de los enunciados siguientes. Si un enunciado es incorrecto, indique porqué.

a) `printf("%-10d\n", 10000);`
b) `printf("%c\n", "This is a string");`
c) `printf("%*.*lf\n", 8, 3, 1024.987654);`
d) `printf("%#o\n%#X\n%#e\n", 17, 17, 1008.83689);`
e) `printf("% ld\n%+ld\n", 1000000, 1000000);`
f) `printf("%10.2E\n", 444.93738);`
g) `printf("%10.2g\n", 444.93738);`
h) `printf("%d\n", 10.987);`

9.6 Encuentre el o los errores en cada uno de los siguientes segmentos de programa. Explique cómo pueden ser corregidos cada uno de ellos.

a) `printf("%s\n", 'Happy Birthday');`
b) `printf("%c\n", 'Hello');`
c) `printf("%c\n", "This is a string");`
d) El siguiente enunciado deberá imprimir `"Bon Voyage"`
```
printf(""%s"", "Bon Voyage");
```
e)
```
char day [] = "Sunday";
printf("%s\n", day [3]);
```
f) `printf('Enter your name: ');`
g) `printf(%f, 123.456);`
h) El siguiente enunciado debería de imprimir los caracteres `'O' y 'K'`.
```
printf("%s%s\n", 'O', 'K');
```
i)
```
char s[10];
scanf("%c", s[7]);
```

9.7 Escriba un programa que cargue el arreglo de 10 elementos `number` con enteros al azar, desde 1 hasta 1000. Para cada uno de los valores, imprima el valor y el total acumulado del número de caracteres impresos. Utilice la especificación de conversión %n para determinar el número de caracteres ya extraídos para cada valor. Imprima el número total de caracteres extraídos para todos los valores incluyendo el valor actual, cada vez que éste sea impreso. La salida deberá tener el formato siguiente:

```
Value       Total characters
342         3
1000        7
963         10
6           11
etc.
```

9.8 Escriba un programa para probar la diferencia entre los especificadores de conversión `%d` y `%i` al ser utilizados en enunciados `scanf`. Utilice los enunciados

```
scanf("%i%d", &x, &y);
printf("%d%d\n", x, y);
```

para introducir e imprimir los valores. Pruebe el programa con los siguientes conjuntos de datos de entrada:

```
 10    10
-10   -10
010   010
0x10  0x10
```

9.9 Escriba un programa que imprima los valores de apuntador utilizando todos los especificadores de conversión enteros y la especificación de conversión `%p`. ¿Cuáles son los que imprimen valores raros? ¿Cuáles son los que causan errores? ¿En cuál de los formatos la especificación de conversión `%p` despliega en su sistema la dirección?

9.10 Escriba un programa para probar el resultado de imprimir el valor entero `12345` y el de punto flotante `1.2345` en varios tamaños de campo. ¿Qué pasa cuando se imprimen los valores en campos que contienen menos dígitos que los valores mismos?

9.11 Escriba un programa que imprima el valor `100.453627` redondeado al dígito, décima, centésima, milésima y decenas de millar más cercano.

9.12 Escriba un programa que desde el teclado introduzca una cadena y determine la longitud de la misma. Imprima la cadena utilizando como ancho de campo dos veces su longitud.

9.13 Escriba un programa que convierta temperaturas Fahrenheit enteras desde `0` hasta `212` grados a temperaturas Celsius de punto flotante con `3` dígitos de precisión. Utilice la fórmula

```
celsius = 5.0 / 9.0 * (fahrenheit - 32);
```

para llevar a cabo el cálculo. La salida deberá ser impresa en dos columnas justificadas a la derecha, cada una de 10 caracteres, y las temperaturas Celsius deberán ser antecedidas por un signo, tanto para valores positivos como negativos.

9.14 Escriba un programa para probar todas las secuencias de escape de la figura 9.16. Para aquellas secuencias de escape que mueven el cursor, imprima un carácter, antes y después de imprimir la secuencia de escape, a fin de que resulte claro dónde se ha movido el cursor.

9.15 Escriba un programa que determine si `?` puede ser impreso como un carácter literal como parte de una cadena de control de formato `printf`, en vez de utilizar la secuencia de escape `\?`

9.16 Escriba un programa que introduzca el valor `437` utilizando cada uno de los especificadores de conversión enteros `scanf`. Imprima cada valor introducido, utilizando todos los especificadores de conversión enteros.

9.17 Escriba un programa que utilice cada uno de los especificadores de conversión `e`, `f` y `g` para introducir el valor `1.2345`. Imprima los valores de cada variable para probar que cada especificador de conversión puede ser utilizado para introducir este mismo valor.

9.18 En algunos lenguajes de programación, las cadenas pueden ser introducidas entre comillas sencillas o dobles. Escriba un programa que lea las tres cadenas `suzy`, `"suzy"`, y `'suzy'`. ¿Son las comillas sencillas y dobles ignoradas o leídas por C, como parte de la cadena?

9.19 Escriba un programa que determine si `?` puede ser impreso como la constante de carácter `'?'`, en vez de la secuencia de escape de carácter constante `'\?'` mediante el uso de especificador de conversión `%c` en la cadena de control de formato de un enunciado `printf`.

9.20 Escriba un programa que utilice el especificador de conversión `g` para extraer el valor `9876.12345`. Imprima el valor con precisiones que vayan desde `1` hasta `9`.