8

Caracteres y cadenas

Objetivos

- Ser capaz de utilizar las funciones de la biblioteca de manejo de caracteres (`ctype`).
- Ser capaz de utilizar las funciones de entrada/salida de cadenas y de caracteres de la biblioteca estándar de entrada/salidas (`stdio`).
- Ser capaz de utilizar las funciones de conversión de cadenas de la biblioteca general de utilerías (`stdlib`).
- Ser capaz de utilizar las funciones de procesamiento de cadenas de la biblioteca de manejo de cadenas (`string`).
- Realizar el poder que tienen las bibliotecas de funciones como medio de conseguir reutilización del software.

El defecto principal de Henry King
Era masticar pequeños pedazos de hilo.
Hilaire Belloc

Adecúe la acción a la palabra, y la palabra a la acción.
William Shakespeare

Un escrito vigoroso es conciso. Una oración no debería contener palabras innecesarias, y un párrafo ninguna oración innecesaria.
William Strunk, Jr.

En una concatenación de acuerdo.
Oliver Goldsmith.

Sinopsis

Resumen • Terminología • Errores comunes de programación • Prácticas sanas de programación • Sugerencias de portabilidad • Ejercicios de autoevaluación • Respuestas a los ejercicios de autoevaluación • Ejercicios • Sección especial: compendio de ejercicios de manipulación de cadenas más avanzados.

8.1 Introducción

En este capítulo, presentamos las funciones estándar de biblioteca de C, que facilitan el procesamiento de cadenas y caracteres. Las funciones le permiten a los programas procesar caracteres, cadenas, líneas de texto y bloques de memoria.

El capítulo analiza las técnicas utilizadas para desarrollar editores, procesadores de palabras, software de disposición de páginas, sistemas de tipografía computarizada, y otros tipos de software de procesamiento de texto. Las manipulaciones de texto con formato ejecutadas por funciones de entrada/salida como **printf** y **scanf** pueden ser puestas en operación, utilizando las funciones analizadas en este capítulo.

8.2 Fundamentos de cadenas y caracteres

Los caracteres son los bloques constructivos fundamentales de los programas fuente. Todo programa está compuesto de una secuencia de caracteres que cuando se agrupa en forma significativa es interpretado por la computadora como una serie de instrucciones utilizadas para llevar a cabo una tarea. Un programa puede contener *constantes de caracteres*. Una *constante de carácter* es un valor **int** representado como un carácter entre comillas sencillas. El valor de una constante de carácter es el valor entero del carácter en el conjunto de caracteres de la máquina. Por ejemplo, **'z'** representa el valor entero de **z**, y **'\n'** representa el valor entero de nueva línea.

Una cadena es una serie de caracteres tratados como una sola unidad. Una cadena puede incluir letras, dígitos y varios *caracteres especiales*, como son **+**, **-**, *****, **/**, **$**, y otros. En C, las *literales de cadena* o *constantes de cadena* se escriben entre dobles comillas, como sigue:

"John Q. Doe"	(un nombre)
"99999 Main Street"	(una dirección de calle)
"Waltham, Massachusetts"	(una ciudad y un estado)
"(201) 555-1212"	(un número telefónico)

En C una cadena es un arreglo de caracteres que termina con el *carácter nulo* (**'\0'**). Se tiene acceso a una cadena mediante un apuntador al primer carácter de la cadena. El valor de una cadena es la dirección de su primer carácter. Por lo tanto, en C, es apropiado decir que una *cadena es un apuntador* —de hecho, un apuntador al primer carácter de una cadena. En este sentido, las cadenas son como los arreglos, porque un arreglo también es un apuntador a su primer elemento.

Una cadena puede ser asignada en una declaración, ya sea a un arreglo de caracteres o a una variable del tipo **char ***. Las declaraciones

```
char color[] = "blue";
char colorPtr = "blue";
```

cada una de ellas inicializa una variable a la cadena **"blue"**. La primera declaración crea un arreglo **color**, de 5 elementos, que contiene los caracteres **'b'**, **'l'**, **'u'**, **'e'** y **'\0'**. La segunda declaración crea la variable de apuntador **colorPtr**, que señala a la cadena **"blue"** en alguna parte de la memoria.

Sugerencia de portabilidad 8.1

Cuando una variable del tipo **char *** *es inicializada con una literal de cadena, algunos compiladores pudieran colocar la cadena en una posición en memoria donde ésta no pueda ser modificada. Si pudiera necesitar modificar una literal de cadena, deberá ser almacenada en un arreglo de caracteres, para asegurarse la modificabilidad en todos los sistemas.*

La declaración de arreglo anterior también podría haberse escrito

```
char color[] = {'b', 'l', 'u', 'e', '\0'};
```

Al declarar un arreglo de caracteres que contenga una cadena, el arreglo debe ser lo suficiente grande para almacenar la cadena y su carácter de terminación **NULL**. La declaración anterior determina el tamaño de la cadena en forma automática, basado en el número de inicializadores en la lista de inicialización.

Error común de programación 8.1

No asignar suficiente espacio en un arreglo de caracteres para almacenar el carácter **NULL** *que da por terminado una cadena.*

Error común de programación 8.2

Imprimir una "cadena" que no contenga un carácter de terminación **NULL**.

Práctica sana de programación 8.1

Al almacenar una cadena de caracteres en un arreglo de caracteres, asegúrese que el arreglo es lo suficientemente grande para contener la cadena más extensa que pueda ser almacenada. C permite que se almacenen cadenas de cualquier longitud. Si una cadena resulta más larga que el arreglo de caracteres en la cual debe almacenarse, los caracteres que excedan del final del arreglo sobreescribirán datos en memoria a continuación del mismo.

Una cadena puede ser asignada a un arreglo utilizando **scanf**. Por ejemplo, el enunciado siguiente asigna una cadena a un arreglo de caracteres **word[20]**.

```
scanf("%s", word);
```

La cadena escrita por el usuario se almacena en **word** (note que **word** es un arreglo que es, naturalmente, un apuntador de tal forma que con el argumento **word** el **&** no se requiere). La función **scanf** leerá caracteres hasta que encuentre un espacio, una nueva línea o un indicador de fin de archivo. Note que la cadena deberá tener una longitud no mayor de 19 caracteres, y así dejar espacio para el carácter de terminación **NULL**. Para que un arreglo de caracteres pueda ser impreso como una cadena, el arreglo debe contener un carácter de terminación **NULL**.

Error común de programación 8.3

Procesar un carácter como si fuera una cadena. Una cadena es un apuntador probablemente un entero respetable grande. Sin embargo, un carácter es un entero pequeño (valores ASCII del rango 0-255). En muchos sistemas, esto causará un error, porque las direcciones bajas de memoria se reservan para fines especiales, como manejadores de interruptores del sistema operativo y, por lo tanto, se incurrirá en "violaciones de acceso".

Error común de programación 8.4

Pasar un carácter como argumento a una función, cuando se espera una cadena.

Error común de programación 8.5

Pasar una cadena como un argumento a una función, cuando se espera un carácter.

8.3 Biblioteca de manejo de caracteres

La biblioteca de manejo de caracteres incluye varias funciones que ejecutan pruebas útiles y manipulaciones de datos de caracteres. Cada función recibe un carácter representado como un **int** o un **EOF** como argumento. Como se analizó en el capítulo 4, a menudo los caracteres son manipulados como enteros, porque en C un carácter es un entero de un byte. Recuerde que **EOF** por lo regular tiene el valor -1 y que algunas arquitecturas de hardware no permiten valores negativos almacenados en variables **char**. Por lo tanto, las funciones de manejo de caracteres manipulan caracteres como enteros. La figura 8.1 resume las funciones de la biblioteca de manejo de caracteres.

Práctica sana de programación 8.2

Al usar funciones de la biblioteca de manejo de caracteres, incluya el archivo de cabecera **<ctype.h>**.

El programa de la figura 8.2 demuestra las funciones ***isdigit***, ***isalpha***, ***isalnum***, *y* ***isxdigit***. La función **isdigit** determina si su argumento es un dígito (**0** al **9**). La función **isalpha** determina si su argumento es una letra en mayúscula (**A** a la **Z**)o una minúscula (**a** a la **z**). La función **isalnum** determina si su argumento es una letra mayúscula, una letra minúscula o un dígito. La función **isxdigit** determina si su argumento es un dígito hexadecimal (**A** a **la F**, **a** a la **f**, **0 al 9**).

El programa de la figura 8.2 utiliza el operador condicional (**?:**) con cada una de las funciones para determinar si en la salida deberá ser impresa la cadena **" is a "** o la cadena **" is not a "** para cada uno de los caracteres probados. Por ejemplo, la expresión

```
isdigit('8') ? "8 is a " : "8 is not a "
```


Prototipo	Descripción de la función
`int isdigit(int c)`	Regresa un valor verdadero si **c** es un dígito, y 0 (falso) de lo contrario.
`int isalpha(int c)`	Si **c** es una letra, regresa un valor verdadero y 0 de lo contrario.
`int isalnum(int c)`	Si **c** es un dígito o una letra, regresa un valor verdadero y 0 de lo contrario.
`int isxdigit(int c)`	Si **c** es un carácter digital hexadecimal, regresa un valor verdadero y 0 de lo contrario. (Vea el Apéndice E, "Sistemas numéricos" para una explicación detallada de los números binarios, octales, decimales y hexadecimales).
`int islower(int c)`	Si **c** es una letra minúscula, regresa un valor verdadero y 0 de lo contrario.
`int isupper(int c)`	Si **c** es una letra mayúscula, regresa un valor verdadero y 0 de lo contrario.
`int tolower(int c)`	Si **c** es una letra mayúscula, **tolower** regresa **c** como una letra minúscula. De lo contrario, **tolower** regresa el argumento sin modificación.
`int toupper(int c)`	Si **c** es una letra minúscula, **toupper** regresa **c** como una letra mayúscula. De lo contrario, **toupper** regresa el argumento sin modificación.
`int isspace(int c)`	Regresa un valor verdadero, si **c** es un carácter de espacio en blanco —nueva línea (**'\n'**), espacio (**' '**), alimentación de forma (**'\f'**), regreso de carro (**'\r'**), tabulador horizontal(**'\t'**), o tabulador vertical (**'\v'**) —y 0 de lo contrario.
`int iscntrl(int c)`	Regresa un valor verdadero si **c** es un carácter de control y 0 de lo contrario.
`int ispunct(int c)`	Regresa un valor verdadero si **c** es un carácter de impresión distinto a un espacio, un dígito, una letra, y 0 de lo contrario.
`int isprint(int c)`	Regresa un valor verdadero si **c** es un carácter de impresión incluyendo el espacio (**' '**), y 0 de lo contrario.
`int isgraph(int c)`	Regresa un valor verdadero si **c** es un carácter de impresión distinto del espacio (**' '**), y 0 de lo contrario.

Fig. 8.1 Resumen de las funciones de biblioteca de manejo de caracteres.

indica que si **'8'** es un dígito, es decir, **isdigit** regresa un valor verdadero (no cero), la cadena **"8 is a "** se imprime, y si **'8'** no es un dígito, es decir **isdigit** regresa **0**, a cadena **"8 is not a "** será impresa.

El programa de la figura 8.3 demuestra las funciones ***islower***, ***isupper***, ***tolower*** y ***toupper***. La función **islower** determina si su argumento es una letra minúscula (**a-z**). La función **isupper** determina si su argumento es una letra mayúscula (**A-Z**). La función **tolower** convierte una letra mayúscula en minúscula, y regresa la letra minúscula. Si el argumento no es una letra mayúscula, **tolower** regresa el argumento sin modificación. La función **toupper** convierte una letra minúscula en mayúscula, y regresa la letra mayúscula. Si el argumento no es una letra minúscula, **toupper** regresa el argumento sin modificación.

```
/* Using functions isdigit, isalpha, isalnum, and isxdigit */
#include <stdio.h>
#include <ctype.h>

main()
{
   printf("%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
       isdigit('8') ? "8 is a " : "8 is not a ", "digit",
       isdigit('#') ? "# is a " : "# is not a ", "digit");
   printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n", "According to isalpha:",
       isalpha('A') ? "A is a " : "A is not a ", "letter",
       isalpha('b') ? "b is a " : "b is not a ", "letter",
       isalpha('&') ? "& is a " : "& is not a ", "letter",
       isalpha('4') ? "4 is a " : "4 is not a ", "letter");
   printf("%s\n%s%s\n%s%s\n%s%s\n\n", "According to isalnum:",
       isalnum('A') ? "A is a " : "A is not a ", "digit or a letter",
       isalnum('8') ? "8 is a " : "8 is not a ", "digit or a letter",
       isalnum('#') ? "# is a " : "# is not a ", "digit or a letter");
   printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n",
       "According to isxdigit:",
       isxdigit('F') ? "F is a " : "F is not a ", "hexadecimal digit",
       isxdigit('J') ? "J is a " : "J is not a ", "hexadecimal digit",
       isxdigit('7') ? "7 is a " : "7 is not a ", "hexadecimal digit",
       isxdigit('$') ? "$ is a " : "$ is not a ", "hexadecimal digit",
       isxdigit('f') ? "f is a " : "f is not a ", "hexadecimal
digit");
   return 0;
}
```

```
According to isdigit:
8 is a digit
# is not digit

According to isalpha:
A is a letter
b is a letter
& is not a letter
4 is not a letter

According to isalnum:
A is a digit or a letter
8 is a digit or a letter
# is not a digit or a letter

According to isxdigit:
F is a hexadecimal digit
J is not a hexadecimal digit
7 is a hexadecimal digit
$ is not a hexadecimal digit
f is not a hexadecimal digit
```

Fig. 8.2 Cómo utilizar **isdigit**, **isalpha**, **isalnum**, e **isxdigit**.


```
/* Using functions islower, isupper, tolower, toupper */
#include <stdio.h>
#include <ctype.h>

main()
{
   printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
          "According to islower:",
          islower('p') ? "p is a " : "p is not a ",
          "lowercase letter",
          islower('P') ? "P is a " : "P is not a ",
          "lowercase letter",
          islower('5') ? "5 is a " : "5 is not a ",
          "lowercase letter",
          islower('!') ? "! is a " : "! is not a ",
          "lowercase letter");
   printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
          "According to isupper:",
          isupper('D') ? "D is an " : "D is not an ",
          "uppercase letter",
          isupper('d') ? "d is an " : "d is not an ",
          "uppercase letter",
          isupper('8') ? "8 is an " : "8 is not an ",
          "uppercase letter",
          isupper('$') ? "$ is an " : "$ is not an ",
          "uppercase letter");
   printf("%s%c\n%s%c\n%s%c\n%s%c\n",
          "u converted to uppercase is ", toupper('u'),
          "7 converted to uppercase is ", toupper('7'),
          "$ converted to uppercase is ", toupper('$'),
          "L converted to lowercase is ", tolower('L'));
   return 0;
}
```

```
According to islower:
p is a lowercase letter
P is not a lowercase letter
5 is not a lowercase letter
! is not a lowercase letter

According to a isupper:
D is an a uppercase letter
d is not an uppercase letter
8 is not an uppercase letter
$ is not an uppercase letter

u converted to uppercase is U
7 converted to uppercase is 7
$ converted to uppercase is $
L converted to lowercase is l
```

Fig. 8.3 Cómo utilizar **islower**, **isupper**, **tolower** y **toupper**.

La figura 8.4 demuestra las funciones ***isspace***, ***iscntrl***, ***ispunct***, ***isprint*** e ***isgraph***. La función **isspace** determina si su argumento es alguno de los siguientes caracteres de espacio en blanco: espacio (' '), alimentación de forma ('\f'), nueva línea ('\n'), retorno de carro ('\r'), tabulador horizontal ('\t'), o tabulador vertical ('\v'). La función **iscntrl** determina si su argumento es alguno de los caracteres de control siguientes: tabulador horizontal ('\t'), tabulador vertical ('\v'), alimentación de forma ('\f'), alerta ('\a'), retroceso ('\b'), retorno de carro ('\r'), o nueva línea ('\n'). La función **ispunct** determina si su argumento es un carácter de impresión distinto de un espacio, de un dígito o de una letra como **$, #, (,), [,], {, }, ;, :,** %, etcétera. La función **isprint** determina si su argumento es un carácter, que puede ser desplegado en pantalla (incluyendo un carácter de espacio). La función **isgraph** prueba los mismos caracteres que **isprint**, sin embargo, el carácter de

```
/* Using functions isspace, iscntrl, ispunct, isprint, isgraph */
#include <stdio.h>
#include <ctype.h>

main()
{
   printf("%s\n%s%s%s\n%s%s%s\n%s%s\n\n", "According to isspace:",
      "Newline", isspace('\n') ? " is a " : " is not a ",
      "whitespace character", "Horizontal tab",
      isspace('\t') ? " is a " : " is not a ",
      "whitespace character",
      isspace('%') ? "% is a " : "% is not a ",
      "whitespace character");
   printf("%s\n%s%s%s\n%s%s\n\n", "According to iscntrl:",
      "Newline", iscntrl('\n') ? " is a " : " is not a ",
      "control character", iscntrl('$') ? "$ is a " : "$ is not a ",
      "control character");
   printf("%s\n%s%s\n%s%s\n%s%s\n\n", "According to ispunct:",
      ispunct(';') ? "; is a " : "; is not a ",
      "punctuation character",
      ispunct('Y') ? "Y is a " : "Y is not a ",
      "punctuation character",
      ispunct('#') ? "# is a " : "# is not a ",
      "punctuation character");
   printf("%s\n%s%s\n%s%s%s\n\n", "According to isprint:",
      isprint('$') ? "$ is a " : "$ is not a ", "printing character",
      "Alert", isprint('\a') ? " is a " : " is not a ",
      "printing character");
   printf("%s\n%s%s\n%s%s%s\n", "According to isgraph:",
      isgraph('Q') ? "Q is a " : "Q is not a ",
      "printing character other than a space",
      "Space", isgraph(' ') ? " is a " : " is not a ",
      "printing character other than a space");
   return 0;
}
```

Fig. 8.4 Cómo utilizar **isspace**, **iscntrl**, **inspunct**, **isprint** e **isgraph** (Parte 1 de 2).


```
According to isspace:
Newline is a whitespace character
Horizontal tab is a whitespace character
% is not a whitespace character

According to iscntrl:
Newline is a control character
$ is not a control character

According to ispunct:
; is a punctuation character
Y is not a punctuation character
# is a punctuation character

According to isprint:
$ is a printing character
Alert is not a printing character

According to isgraph:
Q is a printing character other than a space
Space is not a printing character other than a space
```

Fig. 8.4 Cómo utilizar **isspace**, **iscntrl**, **ispunct**, **isprint** e **isgraph** (Parte 2 de 2).

8.4 Funciones de conversión de cadenas

Esta sección presenta las *funciones de conversión de cadenas* correspondientes a la *biblioteca general de utilerías* (***stdlib***). Estas funciones convierten cadenas de dígitos a enteros y valores de punto flotante. La figura 8.5 resume las funciones de conversión de cadenas. Note la utilización de **const** para declarar la variable **nPtr** en los encabezados de función (léase de derecha a izquierda como "**nPtr** es un apuntador a la constante de carácter"); **const** declara que el valor del argumento no será modificado.

Prototipo de función	Descripción de la función
`double atof(const char *nPtr)`	Convierte la cadena `nPtr` a `double`.
`int atoi(const char *nPtr)`	Convierte la cadena `nPtr` a `int`.
`long atol(const char *nPtr,)`	Convierte la cadena `nPtr` a `int` a largo.
`double strtod(const char *nPtr, char **endPtr)`	Convierte la cadena `nPtr` a double.
`long strtol(const char *nPtr, char **endPtr, int base)`	Convierte la cadena `nPtr` a `long`.
`unsigned long strtoul(const char *nPtr, char **endPtr, int base)`	Convierte la cadena `nPtr` a `unsigned long`.

Fig. 8.5 Resumen de las funciones de conversión de cadenas de la biblioteca general de utilerías.

Práctica sana de programación 8.3

Al utilizar funciones de la biblioteca general de utilerías, incluya el archivo de cabecera `<stdlib.h>`

La función **`atof`** (figura 8.6) convierte su argumento —una cadena que represente un número en punto flotante a un valor **`double`**. La función regresa un valor **`double`**. Si el valor convertido no puede ser representado —por ejemplo, si el primer carácter de la cadena no es un dígito— el comportamiento de la función **`atof`** quedará indefinida.

La función **`atoi`** (figura. 8.7) convierte su argumento —una cadena de dígitos que representa a un entero— a un valor **`int`**. La función regresa un valor **`int`**. Si el valor convertido no puede ser representado, el comportamiento de la función **`atoi`** quedará indefinido.

La función **`atol`** (figura 8.8) convierte su argumento —una cadena de dígitos representando un entero largo— a un valor **`long`**. La función regresa un valor **`long`**. Si el valor convertido no puede ser representado, el comportamiento de la función **`atol`** quedará indefinido. Si **`int`** y **`long`** ambos están almacenados en 4 bytes, la función **`atoi`** y la función **`atol`** funcionan en forma idéntica.

La función **`strtod`** (figura 8.9) convierte una secuencia de caracteres representando un valor en punto flotante a **`double`**. La función recibe dos argumentos —una cadena (**`char*`**) y un apuntador a una cadena. La cadena contiene la secuencia de caracteres a ser convertida a **`double`**. El apuntador se asigna la posición del primer carácter después de la porción convertida de la cadena. El enunciado

```
d = strtod(string, &stringPtr);
```

del programa de la figura 8.9 indica que **`d`** es asignado el valor **`double`** convertido de **`string`**, y **`&stringPtr`** es asignado la posición del primer carácter después del valor convertido (**`51.2`**) en **`string`**.

```
/* Using atof */
#include <stdio.h>
#include <stdlib.h>

main()
{
   double d;

   d = atof("99.0");
   printf("%s%.3f\n%s%.3f\n",
          "The string \"99.0\" converted to double is ", d,
          "The converted value divided by 2 is ", d / 2.0);
   return 0;
}
```

```
The string "99.0" converted to double is 99.000
The converted value divided by 2 is 49.500
```

Fig. 8.6 Cómo utilizar **`atof`**.


```
/* Using atoi */
#include <stdio.h>
#include <stdlib.h>

main()
{
   int i;

   i = atoi("2593");
   printf("%s%d\n%s%d\n",
          "The string \"2593\" converted to int is ", i,
          "The converted value minus 593 is ", i - 593);
   return 0;
}
```

```
The string "2593" converted to int is 2593
The converted value minus 593 is 2000
```

Fig. 8.7 Cómo utilizar **`atoi`**.

```
/* Using atol */
#include <stdio.h>
#include <stdlib.h>

main()
{
   long l;

   l = atol("1000000");
   printf("%s%ld\n%s%ld\n",
       "The string \"1000000\" converted to long int is ", l,
       "The converted value divided by 2 is ", l / 2);
   return 0;
}
```

```
The string "1000000" converted to long int is 1000000
The converted value divided by 2 is 500000
```

Fig. 8.8 Cómo utilizar **`atol`**.

La función **`strtol`** (figura 8.10) convierte a **`long`** una secuencia de caracteres representando un entero. La función recibe tres argumentos —una cadena (**`char *`**), un apuntador a una cadena y un entero. La cadena contiene la secuencia de caracteres a convertirse. El apuntador se asigna la posición del primer carácter después de la porción convertida de la cadena. El entero especifica la *base* del valor bajo conversión. El enunciado

```
x = strtol(string, &remainderPtr, 0);
```

```
/* Using strtod */
#include <stdio.h>
#include <stdlib.h>

main()
{
   double d;
   char *string = "51.2% are admitted";
   char *stringPtr;

   d = strtod(string, &stringPtr);
   printf("The string \"%s\" is converted to the\n",
          string);
   printf("double value %.2f and the string \"%s\"\n",
          d, stringPtr);

   return 0;
}
```

```
The string "51.2% are admitted" is converted to the
double value 51.20 and the string "% are admitted"
```

Fig. 8.9 Cómo utilizar **strtod**.

en el programa de la figura 8.10 se indica que **x** es asignado el valor **long** convertido del **string**. El segundo argumento, **&remainderPtr**, se asigna el resto de **string** después de la conversión. Si se usa **NULL** como segundo argumento se hará que se ignore el resto de la cadena. El tercer argumento, **0**, indica que el valor a convertirse puede ser en base octal (base 8), decimal (base 10), o hexadecimal (base 16). La base puede ser especificada como **0** o cualquier valor entre 2 y 36. Vea el Apéndice E, "Sistemas numéricos", para una explicación detallada de los sistemas octal, decimal y hexadecimal. Las representaciones numéricas de enteros de la base 11 hasta la base 36 utilizan los caracteres A a la Z para representar los valores del 10 al 35. Por ejemplo, los valores hexadecimales pueden consistir de los dígitos 0 al 9 y de los caracteres A a la F. Un entero de base 11 puede consistir de los dígitos 0 al 9 y el carácter A. Un entero de base 24 puede consistir de los dígitos 0-9 y de los caracteres A-N. Un entero en base 36 puede consistir de los dígitos 0 al 9 y de los caracteres A a la Z.

La función ***strtoul*** (figura 8.11) convierte una secuencia de caracteres a **unsigned long** representando un entero **unsigned long**. La función opera en forma idéntica a la función **strtol**. El enunciado

```
x = strtoul(string, &remainderPtr, 0);
```

en el programa de la figura 8.11 indica que **x** es asignado el valor **unsigned long** convertido de **string**. El segundo argumento, **&remainderPtr**, es asignado al resto de **string**, después de la conversión. El tercer argumento, 0, indica que el valor a ser convertido puede estar en formato octal, decimal o hexadecimal.


```
/* Using strtol */
#include <stdio.h>
#include <stdlib.h>

main()
{
   long x;
   char *string = "-1234567abc", *remainderPtr;

   x = strtol(string, &remainderPtr, 0);
   printf("%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",
       "The original string is ", string,
       "The converted value is ", x,
       "The remainder of the original string is ",
       remainderPtr,
       "The converted value plus 567 is ", x + 567);
   return 0;
}
```

```
The original string is "-1234567abc"
The converted value is -1234567
The remainder of the original string is "abc"
The converted value plus 567 is -1234000
```

Fig. 8.10 Cómo utilizar **strtol**.

```
/* Using strtoul */
#include <stdio.h>
#include <stdlib.h>

main()
{
   unsigned long x;
   char *string = "1234567abc", *remainderPtr;

   x = strtoul(string, &remainderPtr, 0);
   printf("%s\"%s\"\n%s%lu\n%s\"%s\"\n%s%lu\n",
          "The original string is ", string,
          "The converted value is ", x,
          "The remainder of the original string is ",
          remainderPtr,
          "The converted value minus 567 is ", x - 567);
   return 0;
}
```

```
The original string is "1234567abc"
The converted value is 1234567
The remainder of the original string is "abc"
The converted value minus 567 is 1234000
```

Fig. 8.11 Cómo utilizar **strtoul**.

8.5 Funciones de la biblioteca estándar de entrada/salida

Esta sección presenta varias funciones de la biblioteca estándar de entrada/salida (**stdio**), específicas para manipular datos de caracteres y de cadenas. La figura 8.12 resume las funciones de entrada y salida de caracteres y de cadenas de la biblioteca estándar de entrada/salidas.

Práctica sana de programación 8.4

Al utilizar funciones de la biblioteca estándar de entrada/salida, incluya el archivo de cabecera **<stdio.h>**.

El programa de la figura 8.13 utiliza la función ***gets*** y ***putchar*** para leer una línea de texto de la entrada estándar (teclado), y sacar en forma recursiva los caracteres de la línea en orden inverso. La función **gets** lee caracteres de la entrada estándar hacia su argumento—un arreglo del tipo **char**— hasta que se encuentra con un carácter de nueva línea o un indicador de fin de archivo. Cuando se termina la lectura, se agrega al arreglo un carácter **NULL** (**'\0'**). La función **putchar** imprime su argumento de carácter. El programa llama la función recursiva **reverse**, para imprimir al revés la línea de texto. Si el primer carácter del arreglo recibido por **reverse** es el carácter nulo (**NULL**) **'\0'** **reverse** regresa. De lo contrario, **reverse** vuelve a ser llamada con la dirección del subarreglo empezando con el elemento **s[1]**, y el carácter **s[0]** es sacado mediante **putchar**, cuando la llamada recursiva ha terminado. El orden de los dos enunciados en la porción **else** de la estructura **if** hace que **reverse** avance hacia el carácter de terminación **NULL** de la cadena, antes de que se imprima un carácter. Conforme se completan las llamadas recursivas, los caracteres son sacados en orden inverso.

El programa de la figura 8.14 utiliza las funciones ***getchar*** y ***puts*** para leer caracteres de la entrada estándar al arreglo de caracteres **sentence**, e imprime el arreglo de caracteres como una cadena. La función **getchar** lee un carácter de la entrada estándar y regresa el carácter como un entero. La función **puts** toma una cadena (**char ***) como argumento, e imprime la cadena seguida por un carácter de nueva línea.

Función prototipo	Descripción de la función
`int getchar(void)`	Introduce el siguiente carácter de la entrada estándar y lo regresa como un entero.
`char *gets(char *s)`	Introduce caracteres de la entrada estándar en el arreglo s hasta que encuentra un carácter de nueva línea o de terminación de archivo. Se agrega al arreglo un carácter de terminación NULL.
`int putchar(int c)`	Imprime el carácter almacenado en c.
`int puts(const char *s)`	Imprime la cadena s seguida por un carácter de nueva línea.
`int sprintf(char *s, const char *format, ...)`	Equivalente a **printf**, excepto que la salida se almacena en el arreglo s, en vez de imprimir a la pantalla.
`int sscanf(char *s, const char *format, ...)`	Equivalente a **scanf**, excepto que la entrada se lee del arreglo s, en vez de leerlo desde el teclado.

Fig. 8.12 Funciones de caracteres y cadenas de la biblioteca estándar de entrada/salida.


```
/* Using gets and putchar */
#include <stdio.h>

main()
{
   char sentence[80];
   void reverse(char *);

   printf("Enter a line of text:\n");
   gets(sentence);

   printf("\nThe line printed backwards is:\n");
   reverse(sentence);

   return 0;
}

void reverse(char *s)
{
   if (s[0] == '\0')
      return;
   else {
      reverse(&s[1]);
      putchar(s[0]);
   }
}
```

```
Enter a line of text:
Characters and Strings

The line printed backwards is:
sgnirtS dna sretcarahC
```

```
Enter a line of text:
able was I ere I saw elba

The line printed backwards is:
able was I ere I saw elba
```

Fig. 8.13 Cómo usar **gets** y **putchar**.

El programa deja de introducir caracteres, cuando **getchar** lee el carácter de nueva línea, escrito por el usuario para terminar la línea de texto. Se agrega una carácter **NULL** al arreglo **sentence**, de tal forma que el arreglo pueda ser tratado como una cadena. La función **puts** imprime la cadena contenida en **sentence**.

```
/* Using getchar and puts */
#include <stdio.h>

main()
{
   char c, sentence[80];
   int i = 0;

   puts("Enter a line of text:");
   while ( ( c = getchar() ) != '\n')
      sentence[i++] = c;

   sentence[i] = '\0';  /* insert NULL at end of string */
   puts("\nThe line entered was:");
   puts(sentence);
   return 0;
}
```

```
Enter a line of text:
This is a test.

The line entered was:
This is a test.
```

Fig. 8.14 Cómo utilizar `getchar` y `puts`.

El programa de la figura 8.15 utiliza la función ***sprintf*** para imprimir datos con formato al arreglo **s** —un arreglo de caracteres. La función usa las mismas especificaciones de conversión que utiliza **printf** (vea el capítulo 9 para un análisis detallado de todas las características de formato de impresión). El programa introduce un valor **int** y un valor **float** al arreglo s para dársele formato y para ser impreso. El arreglo **s** es el primer argumento de **sprintf**.

```
/* Using sprintf */
#include <stdio.h>

main()
{
   char s[80];
   int x;
   float y;

   printf("Enter an integer and a float:\n");
   scanf("%d%f", &x, &y);
   sprintf(s, "Integer:%6d\nFloat:%8.2f", x, y);
   printf("%s\n%s\n",
          "The formatted output stored in array s is:", s);
   return 0;
}
```

Fig. 8.15 Cómo utilizar **sprintf** (parte 1 de 2).


```
Enter an integer and a float:
298  87.375
The formatted output stored in array s is:
Integer:     298
Float:     87.38
```

Fig. 8.15 Cómo utilizar **sprintf** (parte 2 de 2).

El programa de la figura 8.16 utiliza la función ***scanf*** para leer datos con formato del arreglo de caracteres **s**. La función utiliza las mismas especificaciones de conversión que **scanf**. El programa lee un **int** y un **float** del arreglo **s**, y almacena los valores en **x** y en **y**, respectivamente. Los valores de **x** y de **y** son impresos. El arreglo **s** es el primer argumento de **sscanf**.

8.6 Funciones de manipulación de cadenas de la biblioteca de manejo de cadenas

La biblioteca de manejo de cadenas proporciona muchas funciones útiles para manipular datos de cadenas, comparar cadenas, buscar en cadenas caracteres y otras cadenas, dividir cadenas (separar cadenas en partes lógicas) y determinar la longitud de las mismas. Esta sección presenta las funciones de manipulación de cadenas de la biblioteca de manejo de cadenas. Las funciones se resumen en la figura 8.17.

Práctica sana de programación 8.5

Al utilizar funciones de la biblioteca de manejo de cadenas, incluya el archivo de cabecera ***<string.h>*** *.*

```
/* Using sscanf */
#include <stdio.h>

main()
{
   char s[] = "31298 87.375";
   int x;
   float y;

   sscanf(s, "%d%f", &x, &y);
   printf("%s\n%s%6d\n%s%8.3f\n",
          "The values stored in character array s are:",
          "Integer:", x, "Float:", y);
   return 0;
}
```

```
The values stored in character array s are:
Integer: 31298
Float:  87.375
```

Fig. 8.16 Cómo utilizar **sscanf**.

Prototipo de función	Descripción de la función
`char *strcpy(char *s1, const char *s2)`	Copia la cadena `s2` al arreglo `s1`. Es regresado el valor de `s1`.
`char *strncpy(char *s1, const char *s2, size_t n)`	Copia en la mayor parte de `n` caracteres de la cadena `s2` en el arreglo `s1`. Es regresado el valor de `s1`.
`char *strcat(char *s1, const char *s2)`	Agrega la cadena `s2` al arreglo `s1`. El primer carácter de `s2` sobreescribe el carácter de terminación `NULL` de `s1`. Es regresado el valor de `s1`.
`char *strncat(char *s1, const char *s2, size_t n)`	Agrega como máximo `n` caracteres de la cadena `s2` al arreglo `s1`. El primer carácter de `s2` sobreescribe el carácter de terminación `NULL` de `s1`. Es regresado el valor de `s1`.

Fig. 8.17 Funciones de manipulación de cadenas de la biblioteca de manejo de cadenas.

La función ***strcpy*** copia su segundo argumento —una cadena en su primer argumento, un arreglo de caracteres que debe ser lo suficiente grande para almacenar la cadena y su carácter de terminación **NULL**, que también debe ser copiado. La función ***strncpy*** es equivalente a **strcpy**, excepto que **strncpy** especifica el número de caracteres a copiarse de la cadena al arreglo. Note que la función **strncpy** no necesariamente copia el carácter de terminación **NULL** de su segundo argumento. Un carácter de terminación **NULL** será escrito, sólo si el número de caracteres a copiarse es por lo menos uno más que la longitud de la cadena. Por ejemplo, si el segundo argumento es **"test"**, se escribirá un carácter de terminación **NULL** si el tercer argumento a **strncpy** es por lo menos **5** (los 4 caracteres en **"test"** más 1 carácter de terminación **NULL**). Si el tercer argumento es mayor que **5**, al arreglo se agregarán caracteres **NULL**, hasta que quede escrito el número total de caracteres especificado por el tercer argumento.

Error común de programación 8.6

No agregar un carácter de terminación **NULL** *al primer argumento de un* **strncpy**, *cuando el tercer argumento es menor que o igual a la longitud de la cadena del segundo argumento.*

El programa de la figura 8.18 utiliza a **strcpy** para copiar toda la cadena del arreglo **x** al arreglo **y**, y después utiliza **strncpy** para copiar los primeros **14** caracteres del arreglo **x** al arreglo **z**. Se agrega al arreglo **z** un carácter **NULL** (**'\0'**), porque la llamada a **strncpy** en el programa no escribe un carácter de terminación **NULL** (el tercer argumento tiene una longitud menor que la longitud de la cadena del segundo argumento).

La función ***strcat*** agrega su segundo argumento —un cadena— a su primer argumento —un arreglo de caracteres que contiene una cadena. El primer carácter del segundo argumento remplaza el **NULL** (**'\0'**) que termina la cadena del primer argumento. El programador deberá asegurarse que el arreglo utilizado para almacenar la primera cadena es lo suficiente extensa para almacenar dicha primera cadena, la segunda cadena y el carácter de terminación **NULL** (copiado


```
/* Using strcpy and strncpy */
#include <stdio.h>
#include <string.h>

main()
{
   char x[] = "Happy Birthday to You";
   char y[25], z[15];

   printf("%s%s\n%s%s\n",
          "The string in array x is: ", x,
          "The string in array y is: ", strcpy(y, x));

   strncpy(z, x, 14);
   z[14] = '\0';
   printf("The string in array z is: %s\n", z);
   return 0;
}
```

```
The string in array x is: Happy Birthday to You
The string in array y is: Happy Birthday to You
The string in array z is: Happy Birthday
```

Fig. 8.18 Cómo utilizar **strcpy** y **strncpy**.

de la segunda cadena). La función ***strncat*** agrega un número especificado de caracteres de la segunda cadena a la primera cadena. Al resultado de forma automática se le agrega un carácter de terminación **NULL**. El programa de la figura 8.19 demuestra la función **strcat** y la función **strncat**.

```
/* Using strcat and strncat */
#include <stdio.h>
#include <string.h>

main()
{
   char s1[20] = "Happy ";
   char s2[] = "New Year ";
   char s3[40] = "";

   printf("s1 = %s\ns2 = %s\n", s1, s2);
   printf("strcat(s1, s2) = %s\n", strcat(s1, s2));
   printf("strncat(s3, s1, 6) = %s\n", strncat(s3, s1, 6));
   printf("strcat(s3, s1) = %s\n", strcat(s3, s1));
   return 0;
}
```

Fig. 8.19 Cómo utilizar **strcat** y **strncat** (parte 1 de 2).

```
s1 = Happy
s2 = New Year
strcat (s1, s2) = Happy New Year
strncat (s3, s1, 6) = Happy
strcat (s3, s1) = Happy Happy New Year
```

Fig. 8.19 Cómo utilizar `strcat` y `strncat` (parte 2 de 2).

8.7 Funciones de comparación de la biblioteca de manejo de cadenas

En esta sección se estudian las funciones de comparación de cadenas, **strcmp** y **strncmp**, de la biblioteca de manejo de cadenas. En la figura 8.20 aparecen los encabezados de función y una breve descripción de cada una de las funciones.

El programa de la figura 8.21 compara tres cadenas utilizando las funciones ***strcmp*** y ***strncmp***. La función **strcmp** compara su primer argumento de cadena con su segundo argumento de cadena, carácter por carácter. Si las cadenas son iguales la función regresa 0, un valor negativo si la primera cadena es menor que la segunda cadena, y un valor positivo si la primera cadena es mayor que la segunda cadena. La función **strncmp** es equivalente a **strcmp**, excepto que **strncmp** compara sólo hasta un número especificado de caracteres. La función **strncmp** no compara en una cadena caracteres después de un carácter **NULL**. El programa imprime el valor entero regresado por cada llamada de función.

Error común de programación 8.7

Suponiendo que `strcmp` *y* `strncmp` *regresan 1 cuando sus argumentos son iguales. Ambas funciones regresan 0 (valor falso en C) en caso de igualdad. Por lo tanto, cuando se prueben dos cadenas buscando igualdad, para determinar si las cadenas son iguales, el resultado de la función* `strcmp` *o la* `strncmp` *deberá ser comparado con 0.*

Prototipo de función	Descripción de la función
`int strcmp(const char *s1, const char *s2)`	Compara la cadena `s1` con la cadena `s2`. La función regresa 0, menos que 0, o mayor que 0, si `s1` es igual a, menor que o mayor que `s2`, respectivamente.
`int strncmp(const char *s1, const char *s2, size_t n)`	Compara hasta `n` caracteres de la cadena `s1` con la cadena `s2`. La función regresa 0, menos que 0, o mayor que 0, si `s1` es igual a, menor que o mayor que `s2`, respectivamente.

Fig. 8.20 Funciones de comparación de cadenas de la biblioteca de manejo de cadenas.


```
/* Using strcmp and strncmp */
#include <stdio.h>
#include <string.h>

main()
{
   char *s1 = "Happy New Year";
   char *s2 = "Happy New Year";
   char *s3 = "Happy Holidays";

   printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
          "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
          "strcmp(s1, s2) = ", strcmp(s1, s2),
          "strcmp(s1, s3) = ", strcmp(s1, s3),
          "strcmp(s3, s1) = ", strcmp(s3, s1));

   printf("%s%2d\n%s%2d\n%s%2d\n",
          "strncmp(s1, s3, 6) = ", strncmp(s1, s3, 6),
          "strncmp(s1, s3, 7) = ", strncmp(s1, s3, 7),
          "strncmp(s3, s1, 7) = ", strncmp(s3, s1, 7));
   return 0;
}
```

```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Hollidays

strcmp(s1, s2) =  0
strcmp(s1, s3) =  1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) =  0
strncmp(s1, s3, 7) =  1
strncmp(s3, s1, 7) = -1
```

Fig. 8.21 Cómo utilizar `strcmp` y `strncmp`.

Para comprender con exactitud el significado de que una cadena sea "mayor que" o "menor que" otra cadena, considere el proceso de alfabetizar una serie de apellidos. El lector colocaría, sin duda "Jones" antes de "Smith" porque la primera letra de "Jones" viene en el alfabeto antes de la primera letra de "Smith". Pero el alfabeto es más que una lista de 26 letras —es una lista ordenada de caracteres. Cada letra está en posición específica dentro de la lista. "Z" es más que una letra del alfabeto; "Z" específicamente es la veintiseisava letra del alfabeto.

¿Cómo sabe la computadora que una letra en particular viene antes de otra? Todos los caracteres se representan dentro de la computadora como códigos numéricos; cuando la computadora compara dos cadenas, de hecho está comparando los códigos numéricos de los caracteres en las cadenas.

Sugerencia de portabilidad 8.2

Los códigos numéricos internos utilizados para representar caracteres pudieran ser distintos en diferentes computadoras.

En un esfuerzo para estandarizar representaciones de caracteres, la mayor parte de los fabricantes de computadoras han diseñado sus máquinas para utilizar uno de dos esquemas populares de codificación *ASCII* o *EBCDIC*. ASCII significa "American Standard Code for Information Interchange", y EBCDIC significa "Extended Binary Coded Decimal Interchange Code". Existen otros esquemas de codificación, pero éstos son los dos más populares.

ASCII y EBCDIC se conocen como *códigos de caracteres* o *conjuntos de caracteres*. Las manipulaciones de cadenas y de caracteres, de hecho, involucran la manipulación de los códigos numéricos apropiados y no de los caracteres mismos. Esto explica la intercambiabilidad en C entre caracteres y pequeños enteros. Dado que tiene sentido decir que un código numérico es mayor que, menor que o igual a otro código numérico, es posible relacionar varios caracteres o cadenas, una con la otra, refiriéndose a los códigos de caracteres. El Apéndice D contiene una lista de los códigos de caracteres ASCII.

8.8 Funciones de búsqueda de la biblioteca de manejo de cadenas

Esta sección presenta las funciones de biblioteca de manejo de cadenas, que se utilizan para buscar caracteres y otras cadenas, dentro de cadenas. Las funciones se resumen en la figura 8.22. Note que las funciones **strcspn** y **strspn** especifican un regreso del tipo **size_t**. El tipo **size_t** es un tipo definido por la norma como el tipo integral del valor regresado por el operador **sizeof**.

Sugerencia de portabilidad 8.3

El tipo **size_t** *es un sinónimo, dependiente del sistema, para el tipo* **unsigned long** *o para el tipo* **unsigned int**.

Prototipo de función	Descripción de la función
`char *strchr(const char *s, int c)`	Localiza la primera instancia del carácter **c** en la cadena **s**. Si encuentra **c**, regresa un apuntador a **c**. De lo contrario, regresa un apuntador **NULL**.
`size_t strcspn(const char *s1, const char *s2)`	Determina y regresa la longitud del segmento inicial de la cadena **s1**, consistiendo de caracteres no contenidos en la cadena **s2**.
`size_t strspn(const char *s1, const char *s2)`	Determina y regresa la longitud del segmento inicial de la cadena **s1**, que consiste sólo de los caracteres contenidos en la cadena **s2**.

Fig. 8.22 Funciones de manipulación de cadenas de la biblioteca de manejo de cadenas (parte 1 de 2)

Prototipo de función	Descripción de la función
`char *strpbrk(const char *s1, const char *s2)`	Localiza la primera ocurrencia en la cadena **s1** de cualquier carácter de la cadena **s2**. Si encuentra un carácter de la cadena **s2**, regresa un apuntador al carácter en la cadena **s1**. De lo contrario, regresa un apuntador **NULL**.
`char *strrchr(const char *s, int c)`	Localiza la última instancia de **c** en la cadena **s**. Si encuentra **c**, regresa un apuntador a **c** en la cadena **s**. De lo contrario, regresa un apuntador **NULL**.
`char *strstr(const char *s1, const char *s2)`	Localiza la primera ocurrencia en la cadena **s1** de la cadena **s2**. Si la cadena es hallada, regresa un apuntador a la cadena en **s1**. De lo contrario, regresa un apuntador **NULL**.
`char *strtok(char *s1, const char *s2)`	Una secuencia de llamadas a **strtok** divide la cadena **s1** en "tokens" —partes lógicas, como palabras, en una línea de texto— separados por caracteres, contenidos en la cadena **s2**. La primera llamada contiene **s1** como primer argumento, y las llamadas subsecuentes, para continuar esta división de la misma cadena, contienen **NULL** como primer argumento. Para cada llamada se regresa un apuntador al token o ficha actual. Si cuando la función es llamada ya no hay más tokens o fichas, se regresará **NULL**.

Fig. 8.22 Funciones de manipulación de cadenas de la biblioteca de manejo de cadenas (parte 2 de 2)

La función ***strchr*** busca la primera ocurrencia de un carácter en una cadena. Si encuentra dicho carácter, **strchr** regresa un apuntador al carácter en la cadena, de lo contrario **strchr** regresa **NULL**. El programa de la figura 8.23 utiliza **strchr** para buscar las primeras ocurrencias de **'a'** y **'z'** en la cadena **"This is a test"**.

La función ***strcspn*** (figura 8.24) determina la longitud de la parte inicial de la cadena en su primer argumento, que no contenga ningún carácter de la cadena existente en su segundo argumento. La función regresa la longitud del segmento.

La función ***strpbrk*** busca la primera ocurrencia, en su primer argumento de cadena, de cualquier carácter existente en su segundo argumento de cadena. Si encuentra un carácter del segundo argumento, **strpbrk** regresa un apuntador al carácter en el primer argumento, de lo contrario **strpbrk** regresa **NULL**. El programa de la figura 8.25 localiza la primera ocurrencia en **string1**, de cualquier carácter existente en **string2**.

La función ***strrchr*** busca la última ocurrencia de un carácter especificado en una cadena. Si encuentra dicho carácter, **strrchr** regresa un apuntador al carácter en la cadena, de lo contrario **strrchr** regresa **NULL**. El programa de la figura 8.26 busca la última ocurrencia del carácter **'z'** en la cadena **"A zoo has many animals including zebras"**.

```
/* Using strchr */
#include <stdio.h>
#include <string.h>

main()
{
   char *string = "This is a test";
   char character1 = 'a', character2 = 'z';

   if (strchr(string, character1) != NULL)
      printf("\'%c\' was found in \"%s\".\n",
             character1, string);
   else
      printf("\'%c\' was not found in \"%s\".\n",
             character1, string);

   if (strchr(string, character2) != NULL)
      printf("\'%c\' was found in \"%s\".\n",
             character2, string);
   else
      printf("\'%c\' was not found in \"%s\".\n",
             character2, string);
   return 0;
}
```

```
'a' was found in "This is a test".
'z' was not found in "This is a test".
```

Fig. 8.23 Cómo utilizar **strchr**.

```
/* Using strcspn */
#include <stdio.h>
#include <string.h>

main()
{
   char *string1 = "The value is 3.14159";
   char *string2 = "1234567890";

   printf("%s%s\n%s%s\n\n%s\n%s%u",
          "string1 = ", string1, "string2 = ", string2,
          "The length of the initial segment of string1",
          "containing no characters from string2 = ",
          strcspn(string1, string2));
   return 0;
}
```

```
string1 = The value is 3.14159
string2 = 1234567890

The length of the initial segment of string1
containing no characters from string2 = 13
```

Fig. 8.24 Cómo utilizar **strcspn**.


```
/* Using strpbrk */
#include <stdio.h>
#include <string.h>

main()
{
   char *string1 = "This is a test";
   char *string2 = "beware";

   printf("%s\"%s\"\n'%c'%s\n\"%s\"\n",
          "Of the characters in ", string2,
          *strpbrk(string1, string2),
          " is the first character to appear in ", string1);
   return 0;
}
```

```
Of the characters in "beware"
'a' is the first character to appear in
"This is a test"
```

Fig. 8.25 Cómo utilizar **strpbrk**.

```
/* Using strrchr */
#include <stdio.h>
#include <string.h>

main()
{
   char *string1 = "A zoo has many animals including zebras";
   int c = 'z';

   printf("%s\n%s'%c'%s\"%s\"\n",
          "The remainder of string1 beginning with the",
          "last occurrence of character ", c,
          " is: ", strrchr(string1, c));
   return 0;
}
```

```
The remainder of string1 beginning with the
last occurrence of character 'z' is: "zebras"
```

Fig. 8.26 Cómo utilizar **strrchr**.

La función ***strspn*** (figura 8.27) determina la longitud de la parte inicial de la cadena en su primer argumento, que sólo contiene caracteres de la cadena existente en su segundo argumento. La función regresa la longitud del segmento.

```
/* Using strspn */
#include <stdio.h>
#include <string.h>

main()
{
   char *string1 = "The value is 3.14159";
   char *string2 = "aehilsTuv ";

   printf("%s%s\n%s%s\n\n%s\n%s%u\n",
          "string1 = ", string1, "string2 = ", string2,
          "The length of the initial segment of string1",
          "containing only characters from string2 = ",
          strspn(string1, string2));
   return 0;
}
```

```
string1 = The value is 3.14159
string2 = aehilsTuv

The length of the initial segment of string1
containing only characters from string2 = 13
```

Fig. 8.27 Cómo utilizar **strspn**.

La función ***strstr*** busca la primera ocurrencia de su segundo argumento de cadena en su primer argumento de cadena. Si su segunda cadena se encuentra en la primera cadena, se regresa un apuntador a la localización de la cadena en el primer argumento. El programa de la figura 8.28 utiliza **strstr** para encontrar la cadena **"def"** en la cadena **"abcdefabcdef"**.

La función ***strtok*** se utiliza para dividir una cadena en una serie de *tokens*. Un token es una serie de caracteres separados por *caracteres delimitantes* (usualmente espacios o marcas de puntuación). Por ejemplo, en una línea de texto, cada palabra puede ser considerada como un token, y los espacios que separan las palabras pueden ser considerados delimitantes.

Se requieren de múltiples llamadas a **strtok** para dividir una cadena en tokens (suponiendo que la cadena contenga más de uno de ellos). La primera llamada a **strtok** contiene dos argumentos, una cadena que va a ser dividida, y una cadena que contiene los caracteres que separan a los tokens. En el programa de la figura 8.29, el enunciado

```
tokenPtr = strtok(string, " ");
```

asigna **tokenPtr** un apuntador al primer token en **string**. El segundo argumento de **strtok**, **" "**, indica que los tokens en **string** están separados por espacios. La función **strtok** busca el primer carácter en **string** que no sea un carácter delimitante (espacio). Esto iniciará el primer token. La función, a continuación, encuentra el siguiente carácter delimitante dentro de la cadena y lo remplaza con un carácter nulo (**'\0'**). Con esto se da por terminado el token actual. La función **strtok** guarda un apuntador al carácter que sigue al token en **string**, y regresa un apuntador al token o ficha actual.


```
/* Using strstr */
#include <stdio.h>
#include <string.h>

main()
{
   char *string1 = "abcdefabcdef";
   char *string2 = "def";

   printf("%s%s\n%s%s\n\n%s\n%s%s%s\n",
          "string1 = ", string1, "string2 = ", string2,
          "The remainder of string1 beginning with the",
          "first occurrence of string2 is: ",
          strstr(string1, string2));

   return 0;
}
```

```
string1 = abcdefabcdef

string2 = def

The remainder of string1 beginning with the
first occurrence of string2 is: defabcdef
```

Fig. 8.28 Cómo utilizar **strstr**.

```
/* Using strtok */
#include <stdio.h>
#include <string.h>

main()
{
   char string[] = "This is a sentence with 7 tokens";
   char *tokenPtr;

   printf("%s\n%s\n\n%s\n",
          "The string to be tokenized is:", string,
          "The tokens are:");

   tokenPtr = strtok(string, " ");

   while (tokenPtr != NULL) {
      printf("%s\n", tokenPtr);
      tokenPtr = strtok(NULL, " ");
   }

   return 0;
}
```

Fig. 8.29 Cómo utilizar **strtok** (parte 1 de 2).

```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:
This
is
a
sentence
with
7
tokens
```

Fig. 8.29 Cómo utilizar `strtok` (parte 2 de 2).

Las siguientes llamadas a **strtok**, para continuar la división de **string**, contienen a **NULL** como primer argumento. El argumento **NULL** indica que la llamada a **strtok** deberá continuar la división, a partir de la localización en **string**, guardada por la última llamada a **strtok**. Si después de llamar a **strtok** ya no quedan tokens, **strtok** regresará **NULL**. El programa de la figura 8.29 utiliza **strtok** para dividir la cadena **"This is a sentence with 7 tokens"**. Cada token se imprime por separado. Note que **strtok** modifica la cadena de entrada y, por lo tanto, deberá hacerse una copia de la cadena, si después de las llamadas a **strtok** dicha cadena debe ser utilizada otra vez en el programa.

8.9 Funciones de memoria de la biblioteca de manejo de cadenas

Las funciones de la biblioteca de manejo de cadenas presentadas en esta sección, facilitan manipular, comparar y buscar en bloques de memoria. Las funciones tratan los bloques de memoria como arreglos de caracteres. Estas funciones pueden manipular cualquier bloque de datos. En la figura 8.30 se resumen las funciones de memoria de la biblioteca de manejo de cadenas. En los análisis de función, "objeto" se refiere a un bloque de datos.

Los parámetros de apuntador a estas funciones se declaran **void ***. En el capítulo 7, vimos que un apuntador a cualquier tipo de datos puede ser asignado de forma directa a un apuntador del tipo **void ***, y un apuntador del tipo **void *** puede ser asignado directamente a un apuntador de cualquier tipo de datos. Por esta razón, estas funciones pueden recibir apuntadores de cualquier tipo de datos. Debido a que un apuntador **void *** no puede ser desreferenciado, cada función recibe un argumento de tamaño que defina el número de caracteres (bytes) que la función procesará. Por razones de simplicidad, los ejemplos en esta sección manipulan arreglos de caracteres (bloques de caracteres).

La función ***memcpy*** copia un número especificado de caracteres del objeto al cual señala su segundo argumento, al objeto al cual señala su primer argumento. La función puede recibir un apuntador a cualquier tipo de objeto. Si los dos objetos se sobreponen en memoria, es decir, si ambos son parte del mismo objeto, el resultado de esta función queda indefinido. El programa de la figura 8.31 utiliza **memcpy** para copiar la cadena en el arreglo **s2** al arreglo **s1**.

La función ***memmove***, como la función **memcpy**, copia un número especifico de bytes del objeto señalado por su segundo argumento, al objeto señalado por su primer argumento. La copia

Prototipo de función	Descripción de la función
`void *memcpy(void *s1, const void *s2, size_t n)`	Copia **n** caracteres del objeto señalado por **s2** al objeto señalado por **s1**. Regresa un apuntador al objeto resultante.
`void *memmove(void *s1, const void *s2, size_t n)`	Copia **n** caracteres del objeto señalado por **s2** al objeto señalado por **s1**. La copia se ejecuta como si los caracteres primero fueran copiados del objeto al cual apunta **s2** a un arreglo temporal, y a continuación del arreglo temporal al objeto apuntador por **s1**. Regresa un apuntador al objeto resultante.
`int *memcmp(const void *s1, const void *s2, size_t n)`	Compara los primeros **n** caracteres de los objetos señalados por **s1** y **s2**. La función regresa **0**, menos que **0**, o más que **0**, si **s1** es igual a, menor que o mayor que **s2**.
`void *memchr(const void *s, int c, size_t n)`	Localiza la primera instancia de **c** (convertida a **unsigned char**) en los primeros **n** caracteres del objeto señalado por **s**. Si **c** es encontrado, regresa un apuntador a **c** en el objeto. De lo contrario, regresa **NULL**.
`void *memset(void *s, int c, size_t n)`	Copia **c** (convertido a **unsigned char**) en los primeros **n** caracteres del objeto señalado por **s**. Regresa un apuntador hacia el resultado.

Fig. 8.30 Funciones de memoria de la biblioteca de manejo de cadenas.

```
/* Using memcpy */
#include <stdio.h>
#include <string.h>

main()
{
   char s1[17], s2[]  = "Copy this string";

   memcpy(s1, s2, 17);
   printf("%s\n%s\"%s\"\n",
          "After s2 is copied into s1 with memcpy,",
          "s1 contains ", s1);
   return 0;
}
```

```
After s2 is copied into s1 with memcpy,
s1 contains "Copy this string"
```

Fig. 8.31 Cómo utilizar `memcpy`.

se lleva a cabo como si primero los bytes fueran copiados del segundo argumento a un arreglo temporal de caracteres, y a continuación copiados de dicho arreglo temporal al primer argumento. Esto permite que los caracteres de una parte de una cadena sean copiados a otra parte de la misma cadena.

Error común de programación 8.8

Funciones de manipulación de cadenas distintas a **memmove** *que copian caracteres dan resultados indefinidos, cuando la copia se efectúa entre partes de una misma cadena.*

El programa de la figura 8.32 utiliza **memmove** para copiar los últimos **10** bytes del arreglo **x**, en los primeros **10** bytes del arreglo **x**.

La función ***memcmp*** (figura 8.33) compara el número especificado de caracteres de su primer argumento, con los caracteres correspondientes de su segundo argumento. La función regresa un valor mayor que 0, si el primer argumento es mayor que el segundo argumento, regresa 0 si los argumentos son iguales, y regresa un valor menor que 0 si el primer argumento es menor que el segundo argumento.

La función ***memchr*** busca la primera incidencia de un byte, representado como un **unsigned char**, en el número especificado de bytes de un objeto. Si encuentra el byte, regresa un apuntador al byte en el objeto, de lo contrario regresa un apuntador **NULL**. El programa de la figura 8.34 busca el carácter (byte) **'r'** en la cadena **"This is a string"**.

La función **memset** copia el valor del byte en su segundo argumento en un número específico de bytes del objeto al cual señala su primer argumento. El programa en la figura 8.35 utiliza **memset** para copiar **'b'** en los primeros 7 bytes de **string1**.

```
/* Using memmove */
#include <stdio.h>
#include <string.h>

main()
{
   char x[] = "Home Sweet Home";

   printf("%s%s\n",
          "The string in array x before memmove is: ", x);
   printf("%s%s\n",
          "The string in array x after memmove is:  ",
          memmove(x, &x[5], 10));

   return 0;
}
```

```
The string in array x before memmove is: Home Sweet Home
The string in array x after memmove is:  Sweet Home Home
```

Fig. 8.32 Cómo utilizar **memmove**.


```
/* Using memcmp */
#include <stdio.h>
#include <string.h>

main()
{
   char s1[] = "ABCDEFG", s2[] = "ABCDXYZ";

   printf("%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n",
          "s1 = ", s1, "s2 = ", s2,
          "memcmp(s1, s2, 4) = ", memcmp(s1, s2, 4),
          "memcmp(s1, s2, 7) = ", memcmp(s1, s2, 7),
          "memcmp(s2, s1, 7) = ", memcmp(s2, s1, 7));
   return 0;
}
```

```
s1 = ABCDEFG
s2 = ABCDXYZ

memcmp(s1, s2, 4) =  0
memcmp(s1, s2, 7) = -1
memcmp(s2, s1, 7) =  1
```

Fig. 8.33 Cómo utilizar **memcmp**.

```
/* Using memchr */
#include <stdio.h>
#include <string.h>

main()
{
   char *s = "This is a string";

   printf("%s\'%c\'%s\"%s\"\n",
          "The remainder of s after character ", 'r',
          " is found is ", memchr(s, 'r', 16));
   return 0;
}
```

```
The remainder of s after character 'r' is found is "ring"
```

Fig. 8.34 Cómo utilizar **memchr**.

8.10 Otras funciones de la biblioteca de manejo de cadenas

Las dos funciones restantes de la biblioteca de manejo de cadenas son **strerror** y **strlen**. La figura 8.36 resume las funciones **strerror** y **strlen**.

```
/* Using memset */
#include <stdio.h>
#include <string.h>

main()
{
   char string1[15] = "BBBBBBBBBBBBBB";

   printf("string1 = %s\n", string1);
   printf("string1 after memset = %s\n",
          memset(string1, 'b', 7));
   return 0;
}
```

```
string1 = BBBBBBBBBBBBBB
string1 after memset = bbbbbbbBBBBBBB
```

Fig. 8.35 Cómo utilizar **memset**.

La función ***strerror*** toma un número de error y crea un cadena de mensaje de error. Regresa un apuntador a la cadena. El programa de la figura 8.37 demuestra **strerror**.

Sugerencia de portabilidad 8.4

El mensaje generado por **strerror** *es dependiente del sistema.*

La función ***strlen*** toma una cadena como argumento, y regresa el número de caracteres en la cadena —el carácter de terminación no queda incluido en la longitud. El programa de la figura 8.38 demuestra la función **strlen**.

Resumen

- La función **islower** determina si su argumento es una letra minúscula (**a - z**).
- La función **isupper** determina si su argumento es una letra mayúscula (**A - Z**).

Prototipo de función	Descripción de la función
char *strerror(int errornum)	Proyecta **errornum** en una cadena completa de texto, de forma dependiente del sistema. Regresa un apuntador a la cadena.
size_t strlen(const char *s)	Determina la longitud de la cadena **s**. Regresa el número de caracteres que anteceden al carácter de terminación **NULL**.

Fig. 8.36 Funciones de manipulación de cadenas de la biblioteca de manejo de cadenas.


```
/* Using strerror */
#include <stdio.h>
#include <string.h>

main()
{
   printf("%s\n", strerror(2));
   return 0;
}
```

```
Error 2
```

Fig. 8.37 Cómo utilizar **strerror**.

```
/* Using strlen */
#include <stdio.h>
#include <string.h>

main()
{
   char *string1 = "abcdefghijklmnopqrstuvwxyz";
   char *string2 = "four";
   char *string3 = "Boston";

   printf("%s\"%s\"%s%lu\n%s\"%s\"%s%lu\n%s\"%s\"%s%lu\n",
          "The length of ", string1, " is ", strlen(string1),
          "The length of ", string2, " is ", strlen(string2),
          "The length of ", string3, " is ", strlen(string3));
   return 0;
}
```

```
The length of "abcdefghijklmnopqrstuvwxyz" is 26
The length of "four" is 4
The length of "Boston" is 6
```

Fig. 8.38 Cómo utilizar **strlen**.

- La función **isdigit** determina si su argumento es un dígito (**0 - 9**).
- La función **isalpha** determina si su argumento es una letra mayúscula (**A - Z**), o una minúscula (**a - z**).
- La función **isalnum** determina si su argumento es una letra mayúscula (**A - Z**), una minúscula (**a - z**), o un dígito (**0 - 9**).
- La función **isxdigit** determina si su argumento es un dígito hexadecimal (**A - F**, **a - f**, **0 - 9**).

- La función **toupper** convierte una letra minúscula en mayúscula, y regresa la letra mayúscula.
- La función **tolower** convierte una letra mayúscula en minúscula, y regresa la letra minúscula.
- La función **isspace** determina si su argumento es alguno de los caracteres de espacio en blanco siguientes: `' '` (espacio), `'\f'`, `'\n'`, `'\r'`, `'\t'` o bien `'\v'`.
- La función **iscntrl** determina si su argumento es uno de los caracteres de control siguientes: `'\t'`, `'\v'`, `'\f'`, `'\a'`, `'\b'`, `'\r'`, o `'\n'`.
- La función **ispunct** determina si su argumento es un carácter de impresión distinto de un espacio, un dígito o una letra.
- La función **isprint** determina si su argumento es cualquier carácter de impresión, incluye el carácter de espacio.
- La función **isgraph** determina si su argumento es un carácter de impresión distinto al carácter de espacio.
- La función **atof** convierte su argumento —una cadena que empieza con una serie de dígitos que representan un número de punto flotante— a un valor **double**.
- La función **atoi** convierte su argumento —una cadena empezando con una serie de dígitos que representan un entero— a un valor **int**.
- La función **atol** convierte su argumento —una cadena empezando con una serie de dígitos que representan un entero **long**— a un valor **long**.
- La función **strtod** convierte una secuencia de caracteres que representan un valor de punto flotante a **double**. La función recibe dos argumentos —una cadena (**char ***) y un apuntador a **char ***. La cadena contiene la secuencia de caracteres a convertirse, y el apuntador a **char *** se asigna al resto de la cadena, después de la conversión.
- La función **strtol** convierte una secuencia de caracteres representando un entero **long**. La función recibe tres argumentos—una cadena (**char ***), un apuntador a **char ***, y un entero. La cadena contiene la secuencia de caracteres a convertirse, el apuntador a **char *** es asignado el resto de la cadena después de la conversión, y el entero define la base del valor bajo conversión.
- La función **strtoul** convierte una secuencia de caracteres representando un entero a **unsigned long**. La función recibe tres argumentos —una cadena (**char ***), un apuntador a **char ***, y un entero. La cadena contiene la secuencia de caracteres a convertirse, el apuntador a **char *** es asignado el resto de la cadena después de la conversión, y el entero define la base del valor que está siendo convertido.
- La función **gets** lee caracteres de la entrada estándar (teclado) hasta que encuentra un carácter de nueva línea o un indicador de fin de archivo. El argumento a **gets** es un arreglo del tipo **char**. Después de que se termina la lectura, se agrega un carácter **NULL** (`'\0'`) al arreglo.
- La función **putchar** imprime su argumento de carácter.
- La función **getchar** lee un carácter de la entrada estándar y regresa el carácter como un entero. Si encuentra el indicador de fin de archivo, **getchar** regresa **EOF**.
- La función **puts** toma una cadena (**char ***) como argumento e imprime la cadena, seguida por un carácter de nueva línea.

- La función **sprintf** utiliza las mismas especificaciones de conversión que **printf** para imprimir datos con formato, en un arreglo del tipo **char**.
- La función **sscanf** utiliza las mismas especificaciones de conversión que la función **scanf** para leer datos con formato de una cadena.
- La función **strcpy** copia su segundo argumento —una cadena —en su primer argumento— un carácter. El programador deberá asegurarse que el arreglo es lo suficiente grande para almacenar la cadena junto con su carácter de terminación **NULL**.
- La función **strncpy** es equivalente a **strcpy**, excepto que una llamada a **strncpy** especifica el número de caracteres a copiarse de la cadena al arreglo. El carácter de terminación **NULL** sólo será copiado si el número de caracteres a copiarse es uno más que la longitud de la cadena.
- La función **strcat** agrega su segundo argumento de cadena —incluyendo el carácter de terminación **NULL**— a su primer argumento de cadena. El primer carácter de la segunda cadena remplaza el carácter **NULL** (`'\0'`) de la primera cadena. El programador deberá asegurarse que el arreglo utilizado para almacenar la primera cadena es lo suficiente extenso para almacenar tanto la primera como la segunda cadena.
- La función **strncat** agrega un número específico de caracteres provenientes de la segunda a la primera cadena. Al resultado se agrega un carácter de terminación **NULL**.
- La función **strcmp** compara su primer argumento de cadena con su segundo argumento de cadena, carácter por carácter. La función regresa 0 si las cadenas son iguales, regresa un valor negativo si la primera cadena es menor que la segunda cadena, y regresa un valor positivo si la primera cadena es mayor que la segunda cadena.
- La función **strncmp** es equivalente a **strcmp**, excepto que **strncmp** compara un número específico de caracteres. Si el número de caracteres en una de las cadenas es menor que el número de caracteres especificado, **strncmp** comparará caracteres, hasta que se encuentre con el carácter **NULL** en la cadena más corta.
- La función **strchr** busca la primera ocurrencia de un carácter en una cadena. Si encuentra el carácter, **strchr** regresa un apuntador del carácter a la cadena, de lo contrario, **strchr** regresa **NULL**.
- La función **strcspn** determina la longitud de la parte inicial de la cadena de su primer argumento, que no contenga ningún carácter de la cadena incluida en su segundo argumento. La función regresa la longitud del segmento.
- La función **strpbrk** busca en su primer argumento la primera ocurrencia de cualquier carácter existente en su segundo argumento. Si encuentra un carácter del segundo argumento, **strpbrk** regresa un apuntador al carácter, de lo contrario **strpbrk** regresa **NULL**.
- La función **strrchr** busca la última ocurrencia de un carácter en una cadena. Si dicho carácter es encontrado, **strrchr** regresa un apuntador al carácter en la cadena, de lo contrario **strrchr** regresa **NULL**.
- La función **strspn** determina la longitud de la parte inicial de la cadena de su primer argumento, que contenga sólo caracteres de la cadena de su segundo argumento. La función regresa la longitud del segmento.
- La función **strstr** busca la primera ocurrencia de su segundo argumento de cadena, en su primer argumento de cadena. Si la segunda cadena se encuentra en la primera cadena, regresa un apuntador a la posición de la cadena en el primer argumento.

- Una secuencia de llamadas a **strtok** divide la cadena **s1** en tokens, que están separados por caracteres contenidos en la cadena **s2**. La primera llamada contiene a **s1** como primer argumento, y las llamadas subsecuentes para continuar la división de la misma cadena, contienen **NULL** como primer argumento. Para cada llamada se devuelve un apuntador al token actual. Si ya no existen más tokens cuando la función es llamada, regresa un apuntador **NULL**.
- La función **memcpy** copia un número especificado de caracteres del objeto a donde señala su segundo argumento, al objeto a donde señala su primer argumento. La función puede recibir un apuntador a cualquier tipo de objeto. Los apuntadores son recibidos por **memcpy** como apuntadores **void**, y para uso en la función convertidos a apuntadores **char**. La función **memcpy** manipula los bytes del objeto como si fueran caracteres.
- La función **memmove** copia un número especifico de bytes del objeto señalado por su segundo argumento, al objeto señalado por su primer argumento. La copia se lleva a cabo como si los bytes fuesen copiados del segundo argumento a un arreglo de caracteres temporal, y a continuación copiados del arreglo temporal al primer argumento.
- La función **memcmp** compara el número especificado de caracteres en su primero y segundo argumentos.
- La función **memchr** busca la primera señal de un byte, representado como un **unsigned char**, en el número especificado de bytes de un objeto. Si encuentra el byte, regresa un apuntador hacia el byte, de lo contrario, regresa un apuntador **NULL**.
- La función **memset** copia su segundo argumento, tratado como un **unsigned char**, a un número especifico de bytes del objeto al cual señala su primer argumento.
- La función **strerror** define un número entero de error en una cadena completa de texto, de forma dependiente del sistema. Regresa un apuntador hacia la cadena.
- La función **strlen** toma como argumento una cadena, y regresa el número de caracteres en la cadena —en la longitud de la cadena no se incluye el carácter de terminación **NULL**.

Terminología

agregar cadenas a otras cadenas
ASCII
atof
atoi
atol
código de carácter
constante de carácter
conjunto de caracteres
comparar cadenas
carácter de control
copiar cadenas
ctype.h
delimitador
ABCDI
biblioteca general de utilería
getchar
gets
dígitos hexadecimales
isalnum
isalpha
iscntrl
isdigit
isgraph
islower
isprint
ispunct
isspace
isupper

isxdigit
longitud de una cadena
literal
memchr
memcmp
memcpy
memmove
memset
representación del código numérico de un carácter
impresión de carácter
putchar
puts
cadena de búsqueda
sprintf
sscanf
stdio.h
stdlib.h
strcat
strchr
strcmp
strcpy
strcspn
strerror
cadena
funciones de comparación de cadena
concatenación de cadenas
constante de cadena
funciones de conversión de cadenas
string.h
literal de cadena
procesamiento de cadenas
strlen
strncat
strncmp
strncpy
strpbrk
strrchr
strspn
strstr
strtod
strtok
strtol
strtoul
tolower
token
división de cadenas
toupper
caracteres de espacio en blanco
procesamiento de palabras

Errores comunes de programación

8.1 No asignar suficiente espacio en un arreglo de caracteres para almacenar el carácter **NULL** que da por terminado una cadena.

8.2 Imprimir una "cadena" que no contenga un carácter de terminación **NULL**.

8.3 Procesar un carácter como si fuera una cadena. Una cadena es un apuntador —probablemente un entero grande. Sin embargo, un carácter es un entero pequeño (valores ASCII del rango 0-225). En muchos sistemas, esto causará un error, porque las direcciones bajas de memoria se reservan para fines especiales, como manejadores de interruptores del sistema operativo y, por lo tanto, se incurrirá en "violaciones de acceso".

8.4 Pasar un carácter como argumento a una función, cuando se espera una cadena.

8.5 Pasar una cadena como argumento a una función, cuando se espera un carácter.

8.6 No agregar un carácter de terminación **NULL** al primer argumento de un **strncpy**, cuando el tercer argumento es menor que o igual a la longitud de la cadena del segundo argumento.

8.7 Suponiendo que **strcmp** y **strncmp** regresan 1 cuando sus argumentos son iguales. Ambas funciones regresan 0 (valor falso en C) en caso de igualdad. Por lo tanto, cuando se prueben dos cadenas buscando igualdad, para determinar si las cadenas son iguales, el resultado de la función **strcmp** o la **strncmp** deberá ser comparado con 0.

8.8 Funciones de manipulación de cadenas distintas a **memmove** que copian caracteres dan resultados indefinidos, cuando la copia se efectúa entre partes de una misma cadena.

Prácticas sanas de programación

8.1 Al almacenar una cadena de caracteres en un arreglo de caracteres, asegúrese que el arreglo sea lo suficiente extenso para contener la cadena más grande que pueda ser almacenada. C permite que se almacenen cadenas de cualquier longitud. Si una cadena resulta más larga que el arreglo de caracteres en la cual debe almacenarse, los caracteres que excedan del final del arreglo sobreescribirán datos en memoria a continuación del mismo.

8.2 Al usar funciones de la biblioteca de manejo de caracteres, incluya el archivo de cabecera `<ctype.h>`

8.3 Al utilizar funciones de la biblioteca general de utilerías, incluya el archivo de cabecera `<stdlib.h>`

8.4 Al utilizar funciones de la biblioteca estándar de entrada/salidas, incluya el archivo de cabecera `<stdio.h>`.

8.5 Al utilizar funciones de la biblioteca de manejo de cadenas, incluya el archivo de cabecera `<string.h>`.

Sugerencias de portabilidad

8.1 Cuando una variable del tipo `char *` es inicializada con una literal de cadena, algunos compiladores pudieran colocar la cadena en una posición en memoria donde ésta no pueda ser modificada. Si pudiera necesitar modificar una literal de cadena, deberá ser almacenada en un arreglo de caracteres, para asegurarse la modificabilidad en todos los sistemas.

8.2 Los códigos numéricos internos utilizados para representar caracteres pudieran ser distintos en diferentes computadoras.

8.3 El tipo `size_t` es un sinónimo, dependiente del sistema, para el tipo `unsigned long` o para el tipo `unsigned int`.

8.4 El mensaje generado por `strerror` es dependiente del sistema.

Ejercicios de autoevaluación

8.1 Escriba un enunciado para ejecutar cada uno de los siguientes. Suponga que las variables `c` (que almacenan un carácter), `x`, `y` y `z` son del tipo `int`, y las variables `d`, `e`, y `f` son del tipo `float`, que la variable `ptr` es del tipo `char *`, y los arreglos `s1[100]` y `s2[100]` son del tipo `char`.

a) Convierta en una letra mayúscula, el carácter almacenado en la variable `c`. Asigne el resultado a la variable `c`.
b) Determine si el valor de la variable `c` es un dígito. Cuando se despliegue el resultado útilice el operador condicional, para imprimir `"is a "` o `"is not a "`, tal y como se mostró en las figuras 8.2, 8.3 y 8.4.
c) Convierta la cadena `"1234567"` a `long` e imprima el valor.
d) Determine si el valor de la variable `c` es un carácter de control. Cuando se despliegue el resultado utilice el operador condicional para imprimir `"is a "`, o bien `"is not a "`.
e) Lea desde el teclado una línea de texto al arreglo `s1`. No utilice `scanf`.
f) Imprima la línea de texto almacenado en el arreglo `s1`. No utilice `printf`.
g) Asigne `ptr` la posición de la última instancia de `c` en `s1`.
h) Imprima el valor de la variable `c`. No utilice `printf`.
i) Convierta la cadena `"8.63582"` a `double` e imprima el valor.
j) Determine si el valor de `c` es una letra. Cuando se muestre el resultado utilice el operador condicional para imprimir `"is a "`, o bien `"is not a "`.
k) Lea un carácter del teclado, y almacene dicho carácter en la variable `c`.
l) Asigne `ptr` la posición de la primera instancia de `s2` en `s1`.
m) Determine si el valor de la variable `c` es un carácter de impresión. Cuando se muestre el resultado utilice el operador condicional para imprimir `"is a "`, o bien `"is not a "`.
n) Lea tres valores `float` en las variables `d`, `e` o `f` a partir de la cadena `"1.27 10.3 9.432"`.

o) Copie la cadena almacenada en el arreglo `s2` al arreglo `s1`.
p) Asigne `ptr` a la posición en la primera ocurrencia de `s1`, de cualquier carácter de `s2`.
q) Compare la cadena en `s1` con la cadena en `s2`. Imprima el resultado.
r) Asigne `ptr` a la posición en la primera ocurrencia de `c` en `s1`.
s) Utilice `sprintf` para imprimir los valores de las variables enteras `x`, `y` y `z` en el arreglo `s1`. Cada valor deberá ser impreso con un ancho de campo de 7.
t) Agregue 10 caracteres de la cadena en `s2` a la cadena en `s1`.
u) Determine la longitud de la cadena en `s1`. Imprima el resultado.
v) Convierta la cadena `"-21"` a `int` e imprima el valor.
w) Asigne `ptr` a la posición del primer token en `s1`. Los tokens en `s2` están separados por comas (,).

8.2 Muestre dos métodos diferentes para inicializar el arreglo de caracteres `vowel` con la cadena de vocales, `"AEIOU"`

8.3 ¿Qué es lo que, si es que existe, se imprime cuando se ejecutan cada uno de los siguientes enunciados de C? Si el enunciado contiene un error, descríbalo e indique cómo corregirlo. Suponga las siguientes declaraciones de variables:

```
char s1[50] = "jack", s2[50] = "jill", s3[50], *sptr;
```

a) `printf("%c%s", toupper(s1[0]), &s1[1]);`
b) `printf("%s", strcpy(s3, s2));`
c) `printf("%s", strcat(strcat(strcpy(s3, s1),"and"), s2));`
d) `printf("%u", strlen(s1) + strlen(s2));`
e) `printf("%u", strlen(s3));`

8.4 Encuentre el error en cada uno de los segmentos de programas siguientes, y explique cómo corregirlo:

a)
```
char s[10];
strncpy(s, "hello", 5);
printf("%s\n", s);
```
b) `printf("%s", 'a');`
c)
```
char s[12];
strcpy(s, "Welcome Home");
```
d)
```
if (strcmp(string1, string2))
printf("The strings are equal\n");
```

Respuestas a los ejercicios de autoevaluación

8.1
a) `c = toupper(c);`
b)
```
printf("'%c'%sdigit\n",
   c, isdigit(c) ? " is a " : " is not a ");
```
c) `printf("%ld\n", atol("1234567"));`
d)
```
printf("'%c'%scontrol character\n",
   c, iscntrl(c) ? " is a " : " is not a ");
```
e) `gets(s1);`
f) `puts(s1);`
g) `ptr = strrchr(s1, c);`
h) `putchar(c);`
i) `printf("%f\n", atof("8.63582"));`
j)
```
printf("'%c'%sletter\n",
   c, isalpha(c) ? " is a " : " is not a ");
```
k) `c = getchar();`

```
l) ptr = strstr(s1, s2);
m) printf("'%c'%sprinting character\n;
   c, isprint(c) ? " is a " : " is not a ");
n) sscanf("1.27 10.3 0.432", "%f%f%f", &d, &e, &f);
o) strcpy(s1, s2);
p) ptr = strpbrk(s1, s2);
q) printf("strcmp(s1, s2) = %d\n", strcmp(s1, s2));
r) ptr = strchr(s1, c);
s) sprintf(s1, "%7d%7d%7d", x, y, z);
t) strncat(s1, s2, 10);
u) printf("strlen(s1) = %u\n, strlen(s1));
v) printf("%d\n", atoi("-21"));
w) ptr = strtok(s2, ",");
```

8.2
```
char vowel[] = "AEIOU";
CHAR vowel[] = {'A', 'E', 'I', 'O', 'U', '\0'};
```

8.3
a) Jack
b) jill
c) jack and jill
d) 8
e) 13

8.4
a) Error: la función **strncpy** no escribe un carácter de terminación **NULL** al arreglo **s**, porque su tercer argumento es igual a la longitud de la cadena **"hello"**.
   Corrección: haga 6 el tercer argumento de **strncpy**, o asigne **'\0'** a **s[5]**.
b) Error: intentar imprimir una constante de caracteres como una cadena.
   Corrección: utilice **%c** para extraer el carácter, o bien remplace **'a'** con **"a"**.
c) Error: el arreglo de caracteres **s** no tiene suficiente tamaño para almacenar el carácter de terminación **NULL**.
   Corrección: Declare el arreglo con más elementos.
d) Error: la función **strcmp** regresará 0 si las cadenas son iguales, por lo tanto, será falsa la condición en la estructura **if**, y **printf** no será ejecutado.
   Corrección: compare en la condición el resultado de **strcmp** con **0**.

Ejercicios

8.5 Escriba un programa que introduzca un carácter del teclado, y pruebe el carácter con cada una de las funciones en la biblioteca de manejo de caracteres. El programa deberá imprimir el valor regresado por cada una de las funciones.

8.6 Escriba un programa que introduzca una línea de texto con la función **gets**, en el arreglo de caracteres **s[100]**. Extraiga la línea en letras mayúsculas y minúsculas.

8.7 Escriba un programa que introduzca 4 cadenas que representen enteros, convierta las cadenas a enteros, sume los valores e imprima el total de los 4 valores.

8.8 Escriba un programa que introduzca 4 cadenas, que representen valores en punto flotante, convierta las cadenas a valores **double**, sume los valores e imprima el total de los 4 valores.

8.9 Escriba un programa que utilice la función **strcmp**, para comparar dos cadenas introducidas por el usuario. El programa deberá indicar si la primera cadena es menor que, igual que o mayor que la segunda cadena.

8.10 Escriba un programa que utilice la función **strncmp**, para comparar dos cadenas introducidas por el usuario. El programa deberá introducir el número de caracteres a compararse. El programa deberá indicar si la primera cadena es menor que, igual que o mayor que la segunda cadena.

8.11 Escriba un programa que utilice generación de números aleatorios para crear oraciones. El programa deberá utilizar cuatro arreglos de apuntadores a **char** llamados **article, noun, verb,** y **preposition.** El programa deberá crear una oración, seleccionando una palabra al azar de cada uno de los arreglos, en el orden siguiente: **article, noun, verb, preposition, article** y **noun.** Conforme se seleccione cada palabra, deberá ser concatenada con las palabras anteriores en un arreglo lo suficiente extensa para contener a toda la oración. Las palabras deberán estar separadas por espacios. Cuando se extraiga la oración final, deberá iniciar con una letra mayúscula y terminar con un punto. El programa deberá generar 20 oraciones de este tipo.

Los arreglos deberán ser llenados como sigue: el arreglo **article** deberá contener los artículos **"the", "a", "one" "some", y "any"**; el arreglo **noun** deberá contener los nombres **"boy", "girl", "dog", "town",** y **"car"**; el arreglo **verb** deberá contener los verbos **"drove", "jumped", "ran", "walked",** y **"skipped"**; el arreglo **preposition** deberá contener las preposiciones: **"to", "from", "over", "under",** y **"on"**.

Una vez escrito el programa anterior y funcionando, modifíquelo para producir una corta historia que esté formada con varias de estas oraciones. (¡Qué tal le parecería la posibilidad de convertirse en un escritor de tesis al azar!)

8.12 (*Quintillas jocosas*). Una quintilla jocosa es un verso cómico de cinco líneas, en las cuales la primera y segunda línea riman con la quinta, y la tercera rima con la cuarta. Utilizando técnicas similares a las desarrolladas en el ejercicio 8.10, escriba un programa en C que produzca quintillas al azar. Pulir este programa para que pueda producir buenas quintillas es un verdadero reto, ¡pero el resultado puede merecer el esfuerzo!.

8.13 Escriba un programa que cifre frases de la lengua inglesa y las convierta en latín infantil. El latín infantil es una forma de lenguaje codificado utilizado a menudo para diversión. Existen muchas variantes en los métodos utilizados para formar frases en latín infantil. Para simplificar, utilice el algoritmo siguiente:

Para formar una frase en latín infantil a partir de una frase en lengua inglesa, divida la frase en palabras, utilizando la función **strtok**. Para traducir cada palabra inglesa en una palabra en latín infantil, coloque la primera letra de la palabra inglesa al final de la palabra y añada las letras **"ay"**. De ahí la palabra **"jump"** se convierte en **"umpjay"**, la palabra **"the"** se convierte en **"hetay"**, y la palabra **computer** se convierte en **"omputercay"**. Los espacios en blanco entre palabras se conservan como están. Suponga lo siguiente: la frase en inglés está formada de palabras separadas por espacios en blanco, no hay signos de puntuación y todas las palabras tienen dos o más letras. La función **printLatinWord** deberá desplegar cada palabra. *Sugerencia*: cada vez que se encuentre un token en una llamada a **strtok**, pase el apuntador del token a la función **printLatinWord**, e imprima la palabra en latín infantil.

8.14 Escriba un programa que introduzca un número telefónico como cadena, en la forma **(555) 555-5555**. El programa deberá utilizar la función **strtok** para extraer el código de área como token, los tres primeros dígitos del número telefónico como token y los últimos cuatro dígitos del número telefónico como token. Los siete dígitos del número telefónico deberán ser concatenados en una cadena. El programa deberá convertir la cadena del código de área a **int**, y convertir la cadena del número telefónico a **long**. Tanto el código de área como el número telefónico deberán ser impresos.

8.15 Escriba un programa que introduzca una línea de texto, divida la línea con la función **strtok** y extraiga los tokens en orden inverso.

8.16 Escriba un programa que introduzca desde el teclado una línea de texto y una cadena de búsqueda. Utilizando la función **strstr** localice en la línea de texto la primera ocurrencia de la cadena de búsqueda, y asigne la posición a la variable **searchPtr** del tipo **char *.** Si encuentra la cadena de búsqueda, imprima el resto de la línea de texto, empezando con la cadena de texto. A continuación utilice otra vez **strstr**, para localizar en la línea de texto la siguiente ocurrencia de la cadena de búsqueda. Si encuentra una segunda ocurrencia, imprima el resto de la línea de texto, empezando con la segunda ocurrencia. *Sugerencia*: la segunda llamada a **strstr** deberá contener **searchPtr + 1** como primer argumento.

8.17 Escriba un programa basado en el programa del ejercicio 8.16, que introduzca varias líneas de texto y una cadena de búsqueda, y utilice la función `strstr` para determinar todas las ocurrencias de la cadena, en las líneas de texto. Imprima el resultado.

8.18 Escriba un programa que introduzca varias líneas de texto y un carácter de búsqueda, y utilice la función `strchr` para determinar todas las ocurrencias del carácter, en las líneas de texto

8.19 Escriba un programa basado en el programa del ejercicio 8.18 que introduzca varias líneas de texto y utilice la función `strchr` para determinar todas las ocurrencias de cada letra del alfabeto en las líneas de texto. Las letras mayúsculas y minúsculas deberán ser contadas juntas. Almacene en un arreglo los totales de cada letra, y una vez que se hayan determinado los totales, imprímalos en formato tabular.

8.20 Escriba un programa que introduzca varias líneas de texto y utilice `strtok`, para contar el número total de palabras. Suponga que las palabras están separadas, ya sea por espacios o por caracteres de nueva línea.

8.21 Utilice las funciones de comparación de cadenas, analizadas en la Sección 8.6, y las técnicas para ordenamiento de arreglos, desarrolladas en el capítulo 6, para escribir un programa que alfabetice una lista de cadenas. Utilice como datos para su programa los nombres de 10 ó 15 ciudades en su área.

8.22 La gráfica en el apéndice D muestra las representaciones de código numérico, para los caracteres del conjunto de caracteres ASCII. Estudie esta gráfica y a continuación, indique si cada uno de los siguientes es verdadero o falso.

a) La letra `"A"` viene antes de la letra `"B"`.
b) El dígito `"9"` viene antes del dígito `"0"`.
c) Los símbolos comúnmente utilizados para la suma, resta, multiplicación y división, todos ellos vienen antes de cualquiera de los dígitos.
d) Los dígitos vienen antes de las letras.
e) Si un programa de ordenación ordena cadenas en una secuencia ascendente, entonces el programa colocará el símbolo para un paréntesis derecho, antes del símbolo para uno izquierdo.

8.23 Escriba un programa que lea una serie de cadenas, e imprima sólo estas cadenas, empezando con la letra `"b"`.

8.24 Escriba un programa que lea una serie de cadenas, e imprima sólo aquellas cadenas que terminan con las letras `"ED"`.

8.25 Escriba un programa que introduzca un código ASCII e imprima el carácter correspondiente. Modifique este programa de tal forma que genere todos los códigos posibles de tres dígitos en el rango de 000 a 255 e intente imprimir los caracteres correspondientes. ¿Qué es lo que ocurre cuando se ejecuta este programa?

8.26 Utilizando la gráfica de caracteres ASCII del apéndice D como guía, escriba sus propias versiones de las funciones de manejo de caracteres de la figura 8.1.

8.27 Escriba sus propias versiones de las funciones en la figura 8.5 para conversión de cadenas a números.

8.28 Escriba dos versiones de cada una de las funciones de copia y concatenación de cadenas de la figura 8.17. La primera versión deberá utilizar subíndices de arreglos, y la segunda versión deberá utilizar apuntadores y aritmética de apuntadores.

8.29 Escriba sus propias versiones de las funciones `getchar`, `gets`, `putchar`, y `puts`, descritas en la figura 8.12.

8.30 Escriba dos versiones de cada una de las funciones de comparación de cadenas de la figura. 8.20. La primera versión deberá usar subíndices de arreglos, y la segunda apuntadores y aritmética de apuntadores.

8.31 Escriba sus propias versiones de las funciones en la figura 8.22, para la búsqueda de cadenas.

8.32 Escriba su propias versiones de las funciones en la figura 8.30, para manipular bloques de memoria.

8.33 Escriba dos versiones de la función `strlen` de la figura 8.36. La primera versión deberá utilizar subíndices de arreglos, y la segunda deberá utilizar apuntadores y aritmética de apuntadores.

Sección especial: ejercicios avanzados de manipulación de cadenas

Los ejercicios anteriores están relacionados con el texto y diseñados para probar la comprensión del lector sobre conceptos fundamentales de manipulación de cadenas. En esta sección se incluye un conjunto de problemas intermedios y avanzados. El lector encontrará estos problemas excitantes y divertidos. Los problemas varían de forma considerable al grado de dificultad. Algunos requieren de una hora o dos de escritura y puesta en marcha de los programas. Otros son útiles para tareas de laboratorio, que pudieran requerir dos a tres semanas para su estudio y puesta en marcha. Algunos son proyectos de tesis que plantean retos interesantes.

8.34 (*Análisis de texto*). La disponibilidad de computadoras con capacidades de manipulación de cadenas ha resultado en varios métodos interesantes para analizar lo escrito por grandes autores. Se ha puesto gran atención al hecho de saber si William Shakespeare alguna vez existió. Algunos estudiosos creen que existe evidencia sustancial indicando que Christopher Marlowe, de hecho fue el que escribió las obras maestras atribuidas a Shakespeare. Los investigadores han utilizado computadoras para localizar similitudes en los textos de estos dos autores. Este ejercicio examina tres métodos para analizar texto, utilizando una computadora.

a) Escriba un programa que lea varias líneas de texto e imprima una tabla indicando el número de instancias de cada letra del alfabeto en dicho texto. Por ejemplo, la frase

```
To be, or not to be: that is the question:
```

contiene una "a", dos "b", ninguna "c", etcétera.

b) Escriba un programa que lea varias líneas de texto e imprima una tabla que indique el número de palabras de una letra, de dos letras, de tres letras que aparecen en el texto. Por ejemplo, la frase

```
Whether 'tis nobler in the mind to suffer
```

contiene

Longitud de palabra	Ocurrencias
1	0
2	2
3	2
4	2 (including 'tis)
5	0
6	2
7	1

c) Escriba un programa que lea varias líneas de texto e imprima una tabla indicando el número de ocurrencias de cada palabra distinta en el texto. La primera versión de su programa deberá incluir en la tabla las palabras en el mismo orden en que aparecen dentro de texto. Una impresión más interesante (y más útil) deberá intentarse, en la cual las palabras se ordenarán en forma alfabética. Por ejemplo, las líneas

```
To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
```

contiene las palabras "to" tres veces, la palabra "be" dos veces, la palabra "or" una vez, etcétera.

8.35 (*Procesamiento de palabras*). El tratamiento detallado de la manipulación de cadenas en este texto debe atribuirse principalmente al excitante crecimiento, en años recientes, del procesamiento de texto. Una función importante de los sistemas de procesamiento de palabras es el *tipo justificado* —la alineación de las palabras, tanto en los márgenes izquierdo como derecho de una página. Esto genera un documento de apariencia profesional, que parece haber sido formado en tipografía, en vez de preparado en una máquina de escribir. El tipo justificado puede ser llevado a cabo en sistemas de computación, insertando uno o más caracteres en blanco entre cada una de las palabras de una línea, de tal forma que la palabra más a la derecha se alinee con el margen derecho.

Escriba un programa que lea varias líneas de texto e imprima este texto en formato de tipo justificado. Suponga que el texto debe ser impreso en papel de un ancho de 8 1/2 pulgadas, y que se deben dejar márgenes de una pulgada, tanto en el lado derecho como izquierdo de la página impresa. Suponga que la computadora imprime 10 caracteres por cada pulgada horizontal. Por lo tanto, su programa deberá imprimir 6 1/2 pulgadas de texto, es decir, 65 caracteres por línea.

8.36 (*Impresión de fechas en varios formatos*). En la correspondencia de negocios, las fechas se imprimen comúnmente en varios formatos diferentes. Dos de los formatos más comunes son:

```
07/21/55 and July 21, 1955
```

Escriba un programa que lea una fecha en el primer formato y la imprima en el segundo.

8.37 (*Protección de cheques*). Con frecuencia las computadoras se utilizan en sistemas de escritura de cheques, como son aplicaciones de nóminas y cuentas por pagar. Circulan muchas historias raras, en relación con la impresión de nóminas semanales impresas (por error) por cantidades en exceso de 1 millón. Debido a fallas humanas o de máquina, cantidades rarísimas han sido impresas por sistemas computarizados de escritura de cheques. Los diseñadores de estos sistemas, naturalmente, hacen todo tipo de esfuerzo para incorporar controles en sus sistemas, a fin de evitar que se emitan cheques equivocados.

Otro problema serio es la alteración intencional de la cantidad en el cheque, por alguien que intenta cobrar un cheque de forma fraudulenta. Para evitar que sea alterada una cantidad en dólares, la mayor parte de los sistemas computarizados de escritura de cheques emplean una técnica conocida como *protección de cheques*.

Los cheques diseñados para impresión por computadora contienen un número fijo de espacios, en el cual la computadora puede imprimir una cantidad. Suponga que un cheque de nómina contiene ocho espacios en blanco, en el cual la computadora se supone imprimirá la cantidad de la nómina semanal. Si la cantidad es grande, entonces todos los ocho espacios quedarán llenos, por ejemplo:

```
1,230.60   (check amount)
--------
12345678   (position numbers)
```

Por otra parte, si la cantidad es menor de $1000, entonces varios de los espacios quedarían en blanco. Por ejemplo,

```
   99.87
--------
12345678
```

contiene tres espacios en blanco. Si un cheque se imprime con espacios en blanco, es más fácil que alguien pueda alterar la cantidad del cheque. A fin de evitar que se modifique un cheque, muchos sistemas de escritura de cheques *presentan asteriscos*, para proteger la cantidad, como sigue:

```
***99.87
--------
12345678
```


Escriba un programa que introduzca una cantidad en dólares a imprimirse en un cheque, y a continuación imprima en formato protegido de cheque con asteriscos anteriores, si ello fuera necesario. Suponga que para la impresión de una cantidad están disponibles nueve espacios.

8.38 (*Escribir el equivalente en palabras de una cantidad de cheques*). Continuando con el análisis del ejemplo anterior, insistimos en la importancia de diseñar sistemas de escritura de cheques, que impidan la modificación de las cantidades de los cheques. Un método común de seguridad requiere que la cantidad de cheque se escriba tanto en números como en palabras. Aun si alguien es capaz de modificar la cantidad numérica del cheque, resulta en extremo, difícil modificar la cantidad en palabras.

Muchos sistemas computarizados de escritura de cheques no incluyen la cantidad del cheque en palabras. Quizás la razón principal de esta omisión es el hecho de que la mayoría de los lenguajes de alto nivel utilizados en aplicaciones comerciales, no contienen características adecuadas de manipulación de cadenas. Otra razón es que la lógica para la escritura de equivalentes en palabras de las cantidades de los cheques es algo compleja.

Escriba un programa en C que introduzca una cantidad de cheque numérico y escriba el equivalente en palabras de dicha cantidad. Por ejemplo, la cantidad 112.43 deberá quedar escrita como

```
ONE HUNDRED TWELVE and 43/100
```

8.39 (*Código Morse*). Quizás el más famoso de todos los sistemas de codificación es el código Morse, desarrollado por Samuel Morse en 1832, para uso en el sistema telegráfico. El código Morse asigna una serie de puntos y rayas a cada letra del alfabeto, a cada dígito y a unos cuantos caracteres especiales (como el punto, coma, punto y coma; y dos puntos). En los sistemas orientados a sonido, el punto representa un sonido corto y la raya representa un sonido largo. Se utilizan otras representaciones de los puntos y rayas, con sistemas orientados a luz y sistemas de señalización por banderas.

La separación entre palabras se indica por un espacio, o por la ausencia de un punto o de una raya. En un sistema orientado a sonidos, un espacio queda indicado por un corto periodo de tiempo, en el cual ningún sonido se transmite. La versión internacional del código Morse aparece en la figura 8.39.

Escriba un programa que lea una frase en lengua inglesa y que cifre la frase en código Morse. También escriba un programa que lea una frase en código Morse y la convierta en el equivalente en lengua inglesa. Utilice un espacio en blanco entre cada letra codificada Morse y tres espacios en blanco entre cada palabra codificada en Morse.

8.40 (*Programa de conversión métrica*). Escriba un programa que ayude al usuario con las conversiones métricas. Su programa deberá permitir al usuario especificar como cadenas los nombres de las unidades (es decir, centímetros, litros, gramos, etcétera para el sistema métrico y pulgadas, cuartos, libras, etcétera para el sistema inglés) y deberá responder a preguntas sencillas como

```
"¿How many inches are in 2 meters?"
"¿How many liters are in 10 quarts?"
```

Su programa deberá poder reconocer conversiones inválidas. Por ejemplo, la pregunta

```
"¿How many feet in 5 kilograms?"
```

no tiene sentido, porque `"feet"` son unidades de longitud, en tanto que `"kilogram"` es una unidad de peso.

8.41 (*Cartas de cobranza*). Muchos negocios gastan mucho tiempo y dinero cobrando deudas vencidas. La cobranza de cuentas vencidas es el proceso de efectuar demandas repetidas o insistentes por carta a un deudor, en un intento de cobrar una deuda.

A menudo se utilizan computadoras para generar automáticamente cartas de cobranza, en grado creciente de severidad, conforme la cuenta se hace más vieja. La teoría es que mientras más vieja sea la cuenta, más difícil será su cobranza, por lo tanto, las cartas correspondientes deberán ser más y más amenazadoras.

Carácter	Código	Carácter	Código
A	.-	T	-
B	-...	U	..-
C	-.-.	V	...-
D	-..	W	.--
E	.	X	-..-
F	..-.	Y	-.--
G	--.	Z	--..
H	....		
I	..	**Digits**	
J	.---	1	.----
K	-.-	2	..---
L	.-..	3	...--
M	--	4	....-
N	-.	5	.....
O	---	6	-....
P	.--.	7	--...
Q	--.-	8	---..
R	.-.	9	----.
S	...	0	-----

Fig. 8.39 Las letras del alfabeto tal y como se expresan en el código internacional Morse.

Escriba un programa en C que contenga los textos de cinco cartas de cobranza, de severidad creciente. Su programa deberá aceptar como entrada:

1. El nombre del deudor.
2. La dirección del deudor.
3. La cuenta del deudor.
4. La cantidad adeudada.
5. El atraso en la cantidad adeudada (es decir, un mes de vencida, dos meses de vencida, etcétera).

Utilice el atraso en el pago para seleccionar uno de los cinco textos de mensaje, y a continuación imprima la carta de cobranza, insertando donde resulte apropiada la otra información proporcionada por el usuario.

Un proyecto de manipulación de cadenas que resulta un reto

8.42 (*Un generador de palabras cruzadas*). La mayor parte de las personas alguna vez han resuelto un crucigrama, pero pocos han intentado generar uno. Generar un crucigrama es un problema difícil. Se sugiere aquí como un proyecto de manipulación de cadenas, que requiere complejidad y esfuerzo sustancial. Existen muchos ángulos que el programador deberá resolver, para conseguir que funcione, inclusive el más simple

de los programas de generación de crucigramas. Por ejemplo, ¿cómo representa la cuadrícula de un crucigrama en la computadora? ¿Deberá utilizarse una serie de cadenas, o arreglos de doble subíndice? El programador necesitará una fuente de palabras (es decir, un diccionario computarizado) que pueda ser consultado de forma directa por el programa. ¿En qué forma deberán ser almacenadas estas palabras, para facilitar las manipulaciones complejas requeridas por el programa? El lector de verdad ambicioso deseará generar la porción de "indicios o pistas" en la cual se imprimen las breves sugerencias para cada palabra "horizontal" y cada palabra "vertical" para quien está resolviendo el crucigrama. Simplemente imprimir una versión del crucigrama en blanco por sí mismo no resulta un problema sencillo.