Chapter 8
C Characters and Strings

C How to Program, 8/e

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Objectives

In this chapter, you'll:

- Use the functions of the character-handling library (`<ctype.h>`).
- Use the string-conversion functions of the general utilities library (`<stdlib.h>`).
- Use the string and character input/output functions of the standard input/output library (`<stdio.h>`).
- Use the string-processing functions of the string-handling library (`<string.h>`).
- Use the memory-processing functions of the string-handling library (`<string.h>`).

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.1 Introduction

- This chapter introduces the C Standard Library functions that facilitate string and character processing.
- The functions enable programs to process characters, strings, lines of text and blocks of memory.
- The chapter discusses the techniques used to develop editors, word processors, page-layout software, computerized typesetting systems and other kinds of text-processing software.
- The text manipulations performed by formatted input/output functions like `printf` and `scanf` can be implemented using the functions discussed in this chapter.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.2 Fundamentals of Strings and Characters

- Characters are the fundamental building blocks of source programs.
- Every program is composed of a sequence of characters that—when grouped together meaningfully—is interpreted by the computer as a series of instructions used to accomplish a task.
- A program may contain character constants.
- A character constant is an `int` value represented as a character in single quotes.
- The value of a character constant is the integer value of the character in the machine's character set.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.2 Fundamentals of Strings and Characters (Cont.)

- For example, `'z'` represents the integer value of `z`, and `'\n'` the integer value of newline (122 and 10 in ASCII, respectively).
- A string is a series of characters treated as a single unit.
- A string may include letters, digits and various special characters such as +, -, *, / and $.
- String literals, or string constants, in C are written in double quotation marks.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.2 Fundamentals of Strings and Characters (Cont.)

- A string in C is an array of characters ending in the null character (`'\0'`).
- A string is accessed via a *pointer* to the first character in the string.
- The value of a string is the address of its first character.
- Thus, in C, it's appropriate to say that a string is a pointer—in fact, a pointer to the string's first character.
- In this sense, strings are like arrays, because an array is also a pointer to its first element.
- A *character array* or a *variable of type* `char *` can be initialized with a string in a definition.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.2 Fundamentals of Strings and Characters (Cont.)

- The definitions
  - ```
    char color[] = "blue";
    const char *colorPtr = "blue";
    ```

  each initialize a variable to the string `"blue"`.
- The first definition creates a 5-element array `color` containing the characters `'b'`, `'l'`, `'u'`, `'e'` and `'\0'`.
- The second definition creates pointer variable `colorPtr` that points to the string `"blue"` somewhere in memory.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.
    ```

Portability Tip 8.1

The C standard indicates that a a string literal is immutable (i.e., not modifiable), but some compilers do not enforce this. If you might need to modify a string literal, it must be stored in a character array.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.2 Fundamentals of Strings and Characters (Cont.)

- The preceding array definition could also have been written
    - `char color[] = {'b', 'l', 'u', 'e', '\0'};`
- When defining a character array to contain a string, the array must be large enough to store the string *and* its terminating null character.
- The preceding definition automatically determines the size of the array based on the number of initializers in the initializer list.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Common Programming Error 8.1

Not allocating sufficient space in a character array to store the null character that terminates a string is an error.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Common Programming Error 8.2

Printing a "string" that does not contain a terminating null character is an error. Printing will continue past the end of the "string" until a null character is encountered.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Error-Prevention Tip 8.1

When storing a string of characters in a character array, be sure that the array is large enough to hold the largest string that will be stored. C allows strings of any *length to be stored. If a string is longer than the character array in which it's to be stored, characters beyond the end of the array will overwrite data in memory following the array.*

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.2 Fundamentals of Strings and Characters (Cont.)

- A string can be stored in an array using `scanf`.
- For example, the following statement stores a string in character array `word[20]`:
  - `scanf("%s", word);`
- The string entered by the user is stored in `word`.
- Variable `word` is an array, which is, of course, a pointer, so the `&` is not needed with argument `word`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.2 Fundamentals of Strings and Characters (Cont.)

- `scanf` will read characters until a space, tab, newline or end-of-file indicator is encountered.
- So, it's possible that, without the field width `19` in the conversion specifier `%19s`, the user input could exceed 19 characters and that your program might crash!
- For this reason, you should *always* use a field width when using `scanf` to read into a `char` array.
- The field width `19` in the preceding statement ensures that `scanf` reads a *maximum* of 19 characters and saves the last character for the string's terminating null character.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.2 Fundamentals of Strings and Characters (Cont.)

- This prevents `scanf` from writing characters into memory beyond the end of `s`.
- (For reading input lines of arbitrary length, there's a nonstandard—yet widely supported—function `readline`, usually included in `stdio.h`.)
- For a character array to be printed properly as a string, the array must contain a terminating null character.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Common Programming Error 8.3

Processing a single character as a string. A string is a pointer—probably a respectably large integer. However, a character is a small integer (ASCII values range 0–255). On many systems this causes an error, because low memory addresses are reserved for special purposes such as operating-system interrupt handlers—so "access violations" occur.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Common Programming Error 8.4

Passing a character as an argument to a function when a string is expected (and vice versa) is a compilation error.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.3 Character-Handling Library

- The character-handling library (`<ctype.h>`) includes several functions that perform useful tests and manipulations of character data.
- Each function receives an `unsigned char` (represented as an `int`) or `EOF` as an argument.
- `EOF` normally has the value –1.
- Figure 8.1 summarizes the functions of the character-handling library.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Prototype	Function description
`int isblank(int c);`	Returns a true value if `c` is a *blank character* that separates words in a line of text and 0 (false) otherwise. [*Note:* This function is not available in Microsoft Visual C++.]
`int isdigit(int c);`	Returns a true value if `c` is a *digit* and 0 (false) otherwise.
`int isalpha(int c);`	Returns a true value if `c` is a *letter* and 0 (false) otherwise.
`int isalnum(int c);`	Returns a true value if `c` is a *digit* or a *letter* and 0 (false) otherwise.
`int isxdigit(int c);`	Returns a true value if `c` is a *hexadecimal digit character* and 0 (false) otherwise. (See Appendix C for a detailed explanation of binary numbers, octal numbers, decimal numbers and hexadecimal numbers.)
`int islower(int c);`	Returns a true value if `c` is a *lowercase letter* and 0 (false) otherwise.
`int isupper(int c);`	Returns a true value if `c` is an *uppercase letter* and 0 (false) otherwise.
`int tolower(int c);`	If `c` is an *uppercase letter*, `tolower` returns `c` as a *lowercase letter*. Otherwise, `tolower` returns the argument unchanged.
`int toupper(int c);`	If `c` is a *lowercase letter*, `toupper` returns `c` as an *uppercase letter*. Otherwise, `toupper` returns the argument unchanged.

Fig. 8.1 | Character-handling library (`<ctype.h>`) functions. (Part 1 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Prototype	Function description
`int isspace(int c);`	Returns a true value if `c` is a *whitespace character*—newline (`'\n'`), space (`' '`), form feed (`'\f'`), carriage return (`'\r'`), horizontal tab (`'\t'`) or vertical tab (`'\v'`)—and 0 (false) otherwise.
`int iscntrl(int c);`	Returns a true value if `c` is a *control character*—horizontal tab (`'\t'`), vertical tab (`'\v'`), form feed (`'\f'`), alert (`'\a'`), backspace (`'\b'`), carriage return (`'\r'`), newline (`'\n'`) and others—and 0 (false) otherwise.
`int ispunct(int c);`	Returns a true value if `c` is a *printing character other than a space, a digit, or a letter*—such as `$`, `#`, `(`, `)`, `[`, `]`, `{`, `}`, `;`, `:` or `%`—and returns 0 otherwise.
`int isprint(int c);`	Returns a true value if `c` is a *printing character* (i.e., a character that's visible on the screen) *including a space* and returns 0 (false) otherwise.
`int isgraph(int c);`	Returns a true value if `c` is a *printing character other than a space* and returns 0 (false) otherwise.

Fig. 8.1 | Character-handling library (<ctype.h>) functions. (Part 2 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.3.1 Functions `isdigit`, `isalpha`, `isalnum` and `isxdigit`

- Figure 8.2 demonstrates functions `isdigit`, `isalpha`, `isalnum` and `isxdigit`.
- Function `isdigit` determines whether its argument is a digit (`0–9`).
- Function `isalpha` determines whether its argument is an uppercase (`A–Z`) or lowercase letter (`a–z`).
- Function `isalnum` determines whether its argument is an uppercase letter, a lowercase letter or a digit.
- Function `isxdigit` determines whether its argument is a hexadecimal digit (`A–F`, `a–f`, `0–9`).

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.2: fig08_02.c
// Using functions isdigit, isalpha, isalnum, and isxdigit
#include <stdio.h>
#include <ctype.h>

int main(void)
{
   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");

```

Fig. 8.2 | Using functions isdigit, isalpha, isalnum and isxdigit. (Part 1 of 3.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
   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%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");
}
```

Fig. 8.2 | Using functions `isdigit`, `isalpha`, `isalnum` and `isxdigit`. (Part 2 of 3.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

Fig. 8.2 | Using functions `isdigit`, `isalpha`, `isalnum` and `isxdigit`. (Part 3 of 3.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.3.1 Functions `isdigit`, `isalpha`, `isalnum` and `isxdigit` (Cont.)

- Figure 8.2 uses the conditional operator (`?:`) to determine whether the string `" is a "` or the string `" is not a "` should be printed in the output for each character tested.
- For example, the expression
  - `isdigit('8') ? "8 is a " : "8 is not a "`

  indicates that if `'8'` is a digit, the string `"8 is a "` is printed, and if `'8'` is not a digit (i.e., `isdigit` returns `0`), the string `"8 is not a "` is printed.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.3.1 Functions `islower`, `isupper`, `tolower` and `toupper`

- Figure 8.3 demonstrates functions `islower`, `isupper`, `tolower` and `toupper`.
- Function `islower` determines whether its argument is a lowercase letter (`a`–`z`).
- Function `isupper` determines whether its argument is an uppercase letter (`A`–`Z`).
- Function `tolower` converts an uppercase letter to a lowercase letter and returns the lowercase letter.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.3 Character-Handling Library (Cont.)

- If the argument is *not* an uppercase letter, `tolower` returns the argument *unchanged*.
- Function `toupper` converts a lowercase letter to an uppercase letter and returns the uppercase letter.
- If the argument is *not* a lowercase letter, `toupper` returns the argument *unchanged*.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.3: fig08_03.c
// Using functions islower, isupper, tolower and toupper
#include <stdio.h>
#include <ctype.h>

int main(void)
{
   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");

```

Fig. 8.3 | Using functions `islower`, `isupper`, `tolower` and `toupper`. (Part 1 of 3.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
      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'));
   }
```

Fig. 8.3 | Using functions `islower`, `isupper`, `tolower` and `toupper`. (Part 2 of 3.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
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 isupper:
D is an 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 | Using functions `islower`, `isupper`, `tolower` and `toupper`. (Part 3 of 3.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.3.3 Functions `isspace`, `iscntrl`, `ispunct`, `isprint` and `isgraph`

- Figure 8.4 demonstrates functions `isspace`, `iscntrl`, `ispunct`, `isprint` and `isgraph`.
- Function `isspace` determines whether a character is one of the following whitespace characters: space (`' '`), form feed (`'\f'`), newline (`'\n'`), carriage return (`'\r'`), horizontal tab (`'\t'`) or vertical tab (`'\v'`).

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.3.3 Functions `isspace`, `iscntrl`, `ispunct`, `isprint` and `isgraph`

- Function `iscntrl` determines whether a character is one of the following control characters: horizontal tab (`'\t'`), vertical tab (`'\v'`), form feed (`'\f'`), alert (`'\a'`), backspace (`'\b'`), carriage return (`'\r'`) or newline (`'\n'`).
- Function `ispunct` determines if a character is a printing character other than a space, a digit or a letter, such as `$`, `#`, `(`,`)`, `[`,`]`, `{`, `}`, `;`, `:` or `%`.
- Function `isprint` determines whether a character can be displayed on the screen (including the space character).
- Function `isgraph` is the same as `isprint`, except that the space character is not included.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.4: fig08_04.c
// Using functions isspace, iscntrl, ispunct, isprint and isgraph
#include <stdio.h>
#include <ctype.h>

int main(void)
{
   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");

```

Fig. 8.4 | Using functions `isspace`, `iscntrl`, `ispunct`, `isprint` and `isgraph`. (Part 1 of 3.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
    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");
}
```

Fig. 8.4 | Using functions `isspace`, `iscntrl`, `ispunct`, `isprint` and `isgraph`. (Part 2 of 3.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
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 | Using functions `isspace`, `iscntrl`, `ispunct`, `isprint` and `isgraph`. (Part 3 of 3.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.4 String-Conversion Functions

- This section presents the string-conversion functions from the general utilities library (`<stdlib.h>`).
- These functions convert strings of digits to integer and floating-point values.
- Figure 8.5 summarizes the string-conversion functions.
- The C standard also includes `strtoll` and `strtoull` for converting strings to `long long int` and `unsigned long long int`, respectively.
- Note the use of `const` to declare variable `nPtr` in the function headers (read from right to left as "`nPtr` is a pointer to a character constant"); `const` specifies that the argument value will not be modified.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Function prototype	Function description
`double strtod(const char *nPtr, char **endPtr);`	
	Converts the string `nPtr` to `double`.
`long strtol(const char *nPtr, char **endPtr, int base);`	
	Converts the string `nPtr` to `long`.
`unsigned long strtoul(const char *nPtr, char **endPtr, int base);`	
	Converts the string `nPtr` to `unsigned long`.

Fig. 8.5 | String-conversion functions of the general utilities library.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.4.1 Function `strtod`

- Function `strtod` (Fig. 8.6) converts a sequence of characters representing a floating-point value to double.
- The function returns `0` if it’s unable to convert any portion of its first argument to double.
- The function receives two arguments—a string (`char *`) and a pointer to a string (`char **`).
- The string argument contains the character sequence to be converted to double—any whitespace characters at the beginning of the string are ignored.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.4.1 Function `strtod` (Cont.)

- The function uses the `char **` argument to modify a `char *` in the calling function (`stringPtr`) so that it points to the *location of the first character after the converted portion of the string* or to the entire string if no portion can be converted.
- `d = strtod(string, &stringPtr);`

  indicates that `d` is assigned the double value converted from `string`, and `stringPtr` is assigned the location of the first character after the converted value (`51.2`) in `string`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.6: fig08_06.c
// Using function strtod
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   const char *string = "51.2% are admitted"; // initialize string
   char *stringPtr; // create char pointer

   double 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);
}
```

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

Fig. 8.6 | Using function `strtod`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.4.2 Function `strtol`

- Function `strtol` (Fig. 8.7) converts to `long int` a sequence of characters representing an integer.
- The function returns `0` if it's unable to convert any portion of its first argument to `long int`.
- The function receives three arguments—a string (`char *`), a pointer to a string and an integer.
- The string argument contains the character sequence to be converted to `double`—any whitespace characters at the beginning of the string are ignored.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.4.2 Function `strtol` (Cont.)

- The function uses the `char **` argument to modify a `char *` in the calling function (`remainderPtr`) so that it points to the location of the first character after the converted portion of the string or to the entire string if no portion can be converted.
- The integer specifies the base of the value being converted. The line

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

indicates that `x` is assigned the long value converted from string.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.4.2 Function `strtol` (Cont.)

- The second argument, `remainderPtr`, is assigned the remainder of string after the conversion.
- Using `NULL` for the second argument causes the *remainder of the string to be ignored*.
- The third argument, 0, indicates that the value to be converted can be in octal (base 8), decimal (base 10) or hexadecimal (base 16) format. The base can be specified as 0 or any value between 2 and 36.
- Numeric representations of integers from base 11 to base 36 use the characters A–Z to represent the values 10 to 35.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.4.2 Function `strtol` (Cont.)

- For example, hexadecimal values can consist of the digits 0–9 and the characters A–F.
- A base-11 integer can consist of the digits 0–9 and the character A.
- A base-24 integer can consist of the digits 0–9 and the characters A–N.
- A base-36 integer can consist of the digits 0–9 and the characters A–Z.
- The function returns 0 if it's unable to convert any portion of its first argument to a `long int` value.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.7: fig08_07.c
// Using function strtol
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   const char *string = "-1234567abc"; // initialize string pointer
   char *remainderPtr; // create char pointer

   long 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);
}
```

```
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. 8.7 | Using function `strtol`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.4.3 Function `strtoul`

- Function `strtoul` (Fig. 8.8) converts to `unsigned long int` a sequence of characters representing an `unsigned long int` value.
- The function works identically to function `strtol`. The statement

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

in Fig. 8.8 indicates that `x` is assigned the `unsigned long int` value converted from string.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.4.3 Function `strtoul` (Cont.)

- The second argument, `&remainderPtr`, is assigned the remainder of `string` after the conversion.
- The third argument, `0`, indicates that the value to be converted can be in octal, decimal or hexadecimal format.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.8: fig08_08.c
// Using function strtoul
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   const char *string = "1234567abc"; // initialize string pointer
   char *remainderPtr; // create char pointer

   unsigned long int 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);
}
```

```
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. 8.8 | Using function `strtoul`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.5 Standard Input/Output Library Functions

- This section presents several functions from the standard input/output library (`<stdio.h>`) specifically for manipulating character and string data.
- Figure 8.12 summarizes the character and string input/output functions of the standard input/output library.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Function prototype	Function description
`int getchar(void);`	Inputs the next character from the standard input and returns it as an integer.
`char *fgets(char *s, int n, FILE *stream);`	Inputs characters from the specified stream into the array `s` until a *newline* or *end-of-file* character is encountered, or until `n - 1` bytes are read. In this chapter, we specify the stream as `stdin`—the *standard input stream*, which is typically used to read characters from the keyboard. A *terminating null character* is appended to the array. Returns the string that was read into `s`. If a newline is encountered, it's included in the string stored in `s`.
`int putchar(int c);`	Prints the character stored in `c` and returns it as an integer.
`int puts(const char *s);`	Prints the string `s` followed by a *newline* character. Returns a non-zero integer if successful, or `EOF` if an error occurs.

Fig. 8.9 | Standard input/output library character and string functions. (Part 1 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Function prototype	Function description
`int sprintf(char *s, const char *format, ...);`	
	Equivalent to `printf`, except the output is stored in the array `s` instead of printed on the screen. Returns the number of characters written to `s`, or `EOF` if an error occurs. [*Note:* We mention the more secure related functions in the Secure C Programming section of this chapter.]
`int sscanf(char *s, const char *format, ...);`	
	Equivalent to `scanf`, except the input is read from the array `s` rather than from the keyboard. Returns the number of items successfully read by the function, or `EOF` if an error occurs. [*Note:* We mention the more secure related functions in the Secure C Programming section of this chapter.]

Fig. 8.9 | Standard input/output library character and string functions. (Part 2 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.5.1 Functions `fgets` and `putchar`

- Figure 8.10 uses functions `fgets` and `putchar` to read a line of text from the standard input (keyboard) and recursively output the characters of the line in reverse order.
- Function `fgets` reads characters from the *standard input* into its first argument—an array of `char`s—until a *newline* or the *end-of-file* indicator is encountered, or until the maximum number of characters is read.
- The maximum number of characters is one fewer than the value specified in `fgets`'s second argument.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.5.1 Functions `fgets` and `putchar` (Cont.)

- The third argument specifies the *stream* from which to read characters—in this case, we use the *standard input stream* (`stdin`).
- A null character (`'\0'`) is appended to the array when reading terminates.
- Function `putchar` prints its character argument.
- The program calls *recursive* function `reverse` to print the line of text backward.
- If the first character of the array received by `reverse` is the null character `'\0'`, `reverse` returns.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.5.1 Functions `fgets` and `putchar` (Cont.)

- Otherwise, `reverse` is called again with the address of the subarray beginning at element `sPtr[1]`, and character `sPtr[0]` is output with `putchar` when the recursive call is completed.
- The order of the two statements in the `else` portion of the `if` statement causes `reverse` to walk to the terminating null character of the string before a character is printed.
- As the recursive calls are completed, the characters are output in reverse order.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.10: fig08_10.c
// Using functions fgets and putchar
#include <stdio.h>
#define SIZE 80

void reverse(const char * const sPtr); // prototype

int main(void)
{
   char sentence[SIZE]; // create char array

   puts("Enter a line of text:");

   // use fgets to read line of text
   fgets(sentence, SIZE, stdin);

   printf("\n%s", "The line printed backward is:");
   reverse(sentence);
}
```

Fig. 8.10 | Using functions `fgets` and `putchar`. (Part 1 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// recursively outputs characters in string in reverse order
void reverse(const char * const sPtr)
{
   // if end of the string
   if ('\0' == sPtr[0]) { // base case
      return;
   }
   else { // if not end of the string
      reverse(&sPtr[1]); // recursion step
      putchar(sPtr[0]); // use putchar to display character
   }
}
```

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

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

Fig. 8.10 | Using functions `fgets` and `putchar`. (Part 2 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.5.2 Function getchar

- Figure 8.11 uses functions `getchar` and `puts` to read characters from the standard input into character array `sentence` and display the characters as a string.
- Function `getchar` reads a character from the *standard input* and returns the character as an integer.
- Function `puts` takes a string as an argument and displays the string followed by a newline character.
- The program stops inputting characters when either 79 characters have been read or when `getchar` reads the *newline* character entered by the user to end the line.
- A *null character* is appended to array `sentence` so that the array may be treated as a string.
- Then, we use `puts` to display the string contained in `sentence`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.11: fig08_11.c
// Using function getchar.
#include <stdio.h>
#define SIZE 80

int main(void)
{
   int c; // variable to hold character input by user
   char sentence[SIZE]; // create char array
   int i = 0; // initialize counter i

   // prompt user to enter line of text
   puts("Enter a line of text:");

   // use getchar to read each character
   while ((i < SIZE - 1) && (c = getchar()) != '\n') {
      sentence[i++] = c;
   }

   sentence[i] = '\0'; // terminate string

```

Fig. 8.11 | Using function `getchar`. (Part 1 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
   // use puts to display sentence
   puts("\nThe line entered was:");
   puts(sentence);
}
```

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

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

Fig. 8.11 | Using function **getchar**. (Part 2 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.5.3 Function sprintf

- Figure 8.12 uses function `sprintf` to print formatted data into array `s`—an array of characters.
- The function uses the same conversion specifiers as `printf` (see Chapter 9 for a detailed discussion of formatting).
- The program inputs an `int` value and a `double` value to be formatted and printed to array `s`.
- Array `s` is the first argument of `sprintf`.
  – [Note: If your system supports `snprintf_s`, then use that in preference to `sprintf`. If your system doesn't support `snprintf_s` but does support `snprintf`, then use that in preference to `sprintf`. We discuss each of these functions in Appendix F.]

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.12: fig08_12.c
// Using function sprintf
#include <stdio.h>
#define SIZE 80

int main(void)
{
   int x; // x value to be input
   double y; // y value to be input

   puts("Enter an integer and a double:");
   scanf("%d%lf", &x, &y);

   char s[SIZE]; // create char array
   sprintf(s, "integer:%6d\ndouble:%7.2f", x, y);

   printf("%s\n%s\n", "The formatted output stored in array s is:", s);

}
```

Fig. 8.12 | Using function `sprintf`. (Part 1 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

Fig. 8.12 | Using function `sprintf`. (Part 2 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.5.4 Function `sscanf`

- Figure 8.13 uses function `sscanf` to read formatted data from character array `s`.
- The function uses the same conversion specifiers as `scanf`.
- The program reads an `int` and a `double` from array `s` and stores the values in `x` and `y`, respectively.
- The values of `x` and `y` are printed.
- Array `s` is the first argument of `sscanf`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.13: fig08_13.c
// Using function sscanf
#include <stdio.h>

int main(void)
{
   char s[] = "31298 87.375"; // initialize array s
   int x; // x value to be input
   double y; // y value to be input

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

```
The values stored in character array s are:
integer: 31298
double:   87.375
```

Fig. 8.13 | Using function sscanf.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.6 String-Manipulation Functions of the String-Handling Library

- The string-handling library (`<string.h>`) provides many useful functions for manipulating string data (copying strings and concatenating strings), comparing strings, searching strings for characters and other strings, tokenizing strings (separating strings into logical pieces) and determining the length of strings.
- This section presents the string-manipulation functions of the string-handling library.
- The functions are summarized in Fig. 8.17.
- Every function—except for `strncpy`—appends the *null* character to its result.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Function prototype	Function description
`char *strcpy(char *s1, const char *s2)`	
	Copies string `s2` into array `s1`. The value of `s1` is returned.
`char *strncpy(char *s1, const char *s2, size_t n)`	
	Copies at most `n` *characters* of string `s2` into array `s1` and returns `s1`.
`char *strcat(char *s1, const char *s2)`	
	Appends string `s2` to array `s1`. The first character of `s2` *overwrites the terminating null character* of `s1`. The value of `s1` is returned.
`char *strncat(char *s1, const char *s2, size_t n)`	
	Appends at most `n` *characters* of string `s2` to array `s1`. The first character of `s2` *overwrites the terminating null character* of `s1`. The value of `s1` is returned.

Fig. 8.14 | String-manipulation functions of the string-handling library.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.6 String-Manipulation Functions of the String-Handling Library (Cont.)

- Functions `strncpy` and `strncat` specify a parameter of type `size_t`, which is a type defined by the C standard as the integral type of the value returned by operator `sizeof`.
- Function `strcpy` copies its second argument (a string) into its first argument—a character array that *you must ensure is large enough* to store the string and its terminating null character, which is also copied.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.6 String-Manipulation Functions of the String-Handling Library (Cont.)

- Function `strncpy` is equivalent to `strcpy`, except that `strncpy` specifies the number of characters to be copied from the string into the array.
- *Function `strncpy` does not necessarily copy the terminating null character of its second argument.*
- *This occurs only if the number of characters to be copied is at least one more than the length of the string.*

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.6 String-Manipulation Functions of the String-Handling Library (Cont.)

- For example, if `"test"` is the second argument, a terminating null character is written only if the third argument to `strncpy` is at least 5 (four characters in `"test"` plus a terminating null character).
- If the third argument is larger than 5, null characters are appended to the array until the total number of characters specified by the third argument are written.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Common Programming Error 8.5

Not appending a terminating null character to the first argument of a `strncpy` *when the third argument is less than or equal to the length of the string in the second argument.*

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.6.1 Functions `strcpy` and `strncpy`

- Figure 8.15 uses `strcpy` to copy the entire string in array `x` into array `y` and uses `strncpy` to copy the first `14` characters of array `x` into array `z`.
- A *null character* (`'\0'`) is appended to array `z`, because the call to `strncpy` in the program *does not write a terminating null character* (the third argument is less than the string length of the second argument).

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.15: fig08_15.c
// Using functions strcpy and strncpy
#include <stdio.h>
#include <string.h>
#define SIZE1 25
#define SIZE2 15

int main(void)
{
   char x[] = "Happy Birthday to You"; // initialize char array x
   char y[SIZE1]; // create char array y
   char z[SIZE2]; // create char array z

   // copy contents of x into y
   printf("%s%s\n%s%s\n",
      "The string in array x is: ", x,
      "The string in array y is: ", strcpy(y, x));

```

Fig. 8.15 | Using functions `strcpy` and `strncpy`. (Part 1 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
    // copy first 14 characters of x into z. Does not copy null
    // character
    strncpy(z, x, SIZE2 - 1);

    z[SIZE2 - 1] = '\0'; // terminate string in z
    printf("The string in array z is: %s\n", z);
}
```

```
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. 8.15 | Using functions `strcpy` and `strncpy`. (Part 2 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.6.2 Functions `strcat` and `strncat`

- Function `strcat` appends its second argument (a string) to its first argument (a character array containing a string).
- The first character of the second argument replaces the null (`'\0'`) that terminates the string in the first argument.
- *You must ensure that the array used to store the first string is large enough to store the first string, the second string and the terminating null character copied from the second string.*
- Function `strncat` appends a specified number of characters from the second string to the first string.
- A terminating null character is appended to the result.
- Figure 8.16 demonstrates functions `strcat` and `strncat`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.16: fig08_16.c
// Using functions strcat and strncat
#include <stdio.h>
#include <string.h>

int main(void)
{
   char s1[20] = "Happy "; // initialize char array s1
   char s2[] = "New Year "; // initialize char array s2
   char s3[40] = ""; // initialize char array s3 to empty

   printf("s1 = %s\ns2 = %s\n", s1, s2);

   // concatenate s2 to s1
   printf("strcat(s1, s2) = %s\n", strcat(s1, s2) );

   // concatenate first 6 characters of s1 to s3. Place '\0'
   // after last character
   printf("strncat(s3, s1, 6) = %s\n", strncat(s3, s1, 6));

   // concatenate s1 to s3
   printf("strcat(s3, s1) = %s\n", strcat(s3, s1) );
}
```

Fig. 8.16 | Using functions `strcat` and `strncat`. (Part 1 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

Fig. 8.16 | Using functions `strcat` and `strncat`. (Part 2 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.7 Comparison Functions of the String-Handling Library

- This section presents the string-handling library's string-comparison functions, `strcmp` and `strncmp`.
- Fig. 8.17 contains their prototypes and a brief description of each function.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Function prototype	Function description
`int strcmp(const char *s1, const char *s2);`	
	Compares the string `s1` with the string `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`, respectively.
`int strncmp(const char *s1, const char *s2, size_t n);`	
	Compares up to `n` *characters* of the string `s1` with the string `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`, respectively.

Fig. 8.17 | String-comparison functions of the string-handling library.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.7 Comparison Functions of the String-Handling Library (Cont.)

- Figure 8.18 compares three strings using `strcmp` and `strncmp`.
- Function `strcmp` compares its first string argument with its second string argument, character by character.
- The function returns 0 if the strings are equal, a *negative value* if the first string is less than the second string and a *positive value* if the first string is greater than the second string.
- Function `strncmp` is equivalent to `strcmp`, except that `strncmp` compares up to a specified number of characters.
- Function `strncmp` does *not* compare characters following a null character in a string.
- The program prints the integer value returned by each function call.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.18: fig08_18.c
// Using functions strcmp and strncmp
#include <stdio.h>
#include <string.h>

int main(void)
{
   const char *s1 = "Happy New Year"; // initialize char pointer
   const char *s2 = "Happy New Year"; // initialize char pointer
   const char *s3 = "Happy Holidays"; // initialize char pointer

   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) );
}
```

Fig. 8.18 | Using functions `strcmp` and `strncmp`. (Part 1 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

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. 8.18 | Using functions `strcmp` and `strncmp`. (Part 2 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Common Programming Error 8.6

Assuming that `strcmp` *and* `strncmp` *return 1 when their arguments are equal is a logic error. Both functions return 0 (strangely, the equivalent of C's false value) for equality. Therefore, when comparing two strings for equality, the result of function* `strcmp` *or* `strncmp` *should be compared with 0 to determine whether the strings are equal.*

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.7 Comparison Functions of the String-Handling Library (Cont.)

- To understand just what it means for one string to be "greater than" or "less than" another, consider the process of alphabetizing a series of last names.
- The reader would, no doubt, place "Jones" before "Smith," because the first letter of "Jones" comes before the first letter of "Smith" in the alphabet.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.7 Comparison Functions of the String-Handling Library (Cont.)

- But the alphabet is more than just a list of 26 letters—it's an ordered list of characters.
- Each letter occurs in a specific position within the list.
- "Z" is more than merely a letter of the alphabet; "Z" is specifically the 26th letter of the alphabet.
- How do the string comparison functions know that one particular letter comes before another?
- All characters are represented inside the computer as numeric codes in character sets such as ASCII and Unicode; when the computer compares two strings, it actually compares the numeric codes of the characters in the strings.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.8 Search Functions of the String-Handling Library

- This section presents the functions of the string-handling library used to search strings for characters and other strings.
- The functions are summarized in Fig. 8.19.
- The functions `strcspn` and `strspn` return `size_t`.
- [*Note*: Function `strtok` has a more secure version described in optional Annex K of the C11 standard. We mention this in the Secure C Programming section of this chapter and in Appendix F.]

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Function prototypes and descriptions
`char *strchr(const char *s, int c);`
Locates the first occurrence of character `c` in string `s`. If `c` is found, a pointer to `c` in `s` is returned. Otherwise, a `NULL` pointer is returned.
`size_t strcspn(const char *s1, const char *s2);`
Determines and returns the length of the initial segment of string `s1` consisting of characters *not* contained in string `s2`.
`size_t strspn(const char *s1, const char *s2);`
Determines and returns the length of the initial segment of string `s1` consisting *only* of characters contained in string `s2`.
`char *strpbrk(const char *s1, const char *s2);`
Locates the first occurrence in string `s1` of any character in string `s2`. If a character from string `s2` is found, a pointer to the character in string `s1` is returned. Otherwise, a `NULL` pointer is returned.
`char *strrchr(const char *s, int c);`
Locates the last occurrence of `c` in string `s`. If `c` is found, a pointer to `c` in string `s` is returned. Otherwise, a `NULL` pointer is returned.

Fig. 8.19 | Search functions of the string-handling library. (Part 1 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Function prototypes and descriptions
`char *strstr(const char *s1, const char *s2);` *Locates the first occurrence* in string `s1` of string `s2`. If the string is found, a pointer to the string in `s1` is returned. Otherwise, a `NULL` pointer is returned.
`char *strtok(char *s1, const char *s2);` A sequence of calls to `strtok` breaks string `s1` into *tokens*—logical pieces such as words in a line of text—separated by characters contained in string `s2`. The first call contains `s1` as the first argument, and subsequent calls to continue tokenizing the same string contain `NULL` as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, `NULL` is returned.

Fig. 8.19 | Search functions of the string-handling library. (Part 2 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.8.1 Function `strchr`

- Function `strchr` searches for the *first occurrence* of a character in a string.
- If the character is found, `strchr` returns a pointer to the character in the string; otherwise, `strchr` returns `NULL`.
- Figure 8.20 searches for the first occurrences of `'a'` and `'z'` in `"This is a test"`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.20: fig08_20.c
// Using function strchr
#include <stdio.h>
#include <string.h>

int main(void)
{
   const char *string = "This is a test"; // initialize char pointer
   char character1 = 'a'; // initialize character1
   char character2 = 'z'; // initialize character2

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

```

Fig. 8.20 | Using function `strchr`. (Part 1 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

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

Fig. 8.20 | Using function `strchr`. (Part 2 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.8.2 Function `strcspn`

- Function `strcspn` (Fig. 8.21) determines the length of the initial part of the string in its first argument that does *not* contain any characters from the string in its second argument.
- The function returns the length of the segment.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.21: fig08_21.c
// Using function strcspn
#include <stdio.h>
#include <string.h>

int main(void)
{
   // initialize two char pointers
   const char *string1 = "The value is 3.14159";
   const char *string2 = "1234567890";

   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 no characters from string2 = ",
      strcspn(string1, string2) );
}
```

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

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

Fig. 8.21 | Using function strcspn.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.8.3 Function `strpbrk`

- Function `strpbrk` searches its first string argument for the *first occurrence* of any character in its second string argument.
- If a character from the second argument is found, `strpbrk` returns a pointer to the character in the first argument; otherwise, `strpbrk` returns `NULL`.
- Figure 8.22 shows a program that locates the first occurrence in `string1` of any character from `string2`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.22: fig08_22.c
// Using function strpbrk
#include <stdio.h>
#include <string.h>

int main(void)
{
   const char *string1 = "This is a test"; // initialize char pointer
   const char *string2 = "beware"; // initialize char pointer

   printf("%s\"%s\"\n'%c'%s\n\"%s\"\n",
      "Of the characters in ", string2,
      *strpbrk(string1, string2) ,
      " appears earliest in ", string1);
}
```

```
Of the characters in "beware"
'a' appears earliest in
"This is a test"
```

Fig. 8.22 | Using function strpbrk.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.8.4 Function `strrchr`

- Function `strrchr` searches for the *last occurrence* of the specified character in a string.
- If the character is found, `strrchr` returns a pointer to the character in the string; otherwise, `strrchr` returns `NULL`.
- Figure 8.23 shows a program that searches for the last occurrence of the character `'z'` in the string "`A zoo has many animals including zebras.`"

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.23: fig08_23.c
// Using function strrchr
#include <stdio.h>
#include <string.h>

int main(void)
{
   // initialize char pointer
   const char *string1 = "A zoo has many animals including zebras";

   int c = 'z'; // character to search for

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

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

Fig. 8.23 | Using function `strrchr`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.8.5 Function `strspn`

- Function `strspn` (Fig. 8.24) determines the length of the *initial part* of the string in its first argument that contains only characters from the string in its second argument.
- The function returns the length of the segment.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.24: fig08_24.c
// Using function strspn
#include <stdio.h>
#include <string.h>

int main(void)
{
   // initialize two char pointers
   const char *string1 = "The value is 3.14159";
   const char *string2 = "aehi lsTuv";

   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));
}
```

```
string1 = The value is 3.14159
string2 = aehi lsTuv

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

Fig. 8.24 | Using function strspn.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.8.6 Function `strstr`

- Function `strstr` searches for the *first occurrence* of its second string argument in its first string argument.
- If the second string is found in the first string, a pointer to the location of the string in the first argument is returned.
- Figure 8.25 uses `strstr` to find the string `"def"` in the string `"abcdefabcdef"`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.25: fig08_25.c
// Using function strstr
#include <stdio.h>
#include <string.h>

int main(void)
{
   const char *string1 = "abcdefabcdef"; // string to search
   const char *string2 = "def"; // string to search for

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

```
string1 = abcdefabcdef
string2 = def

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

Fig. 8.25 | Using function `strstr`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.8.7 Function strtok

- Function `strtok` (Fig. 8.26) is used to break a string into a series of tokens.
- A token is a sequence of characters separated by delimiters (usually *spaces* or *punctuation marks*, but a delimiter can be *any character*).
- For example, in a line of text, each word can be considered a token, and the spaces and punctuation separating the words can be considered delimiters.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.26: fig08_26.c
// Using function strtok
#include <stdio.h>
#include <string.h>

int main(void)
{
   // initialize array string
   char string[] = "This is a sentence with 7 tokens";

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

   char *tokenPtr = strtok(string, " "); // begin tokenizing sentence

   // continue tokenizing sentence until tokenPtr becomes NULL
   while (tokenPtr != NULL) {
      printf("%s\n", tokenPtr);
      tokenPtr = strtok(NULL, " "); // get next token
   }
}
```

Fig. 8.26 | Using function `strtok`. (Part 1 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

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

Fig. 8.26 | Using function `strtok`. (Part 2 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.8.7 Function strtok (Cont.)

- Multiple calls to `strtok` are required to tokenize a string—i.e., break it into tokens (assuming that the string contains more than one token).
- The first call to `strtok` contains two arguments: a string to be tokenized, and a string containing characters that separate the tokens.
- The statement
  - ```
    // begin tokenizing sentence
    tokenPtr = strtok(string, " ");
    ```

  assigns `tokenPtr` a pointer to the first token in `string`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.
    ```

8.8.7 Function strtok (Cont.)

- The second argument, `" "`, indicates that tokens are separated by spaces.
- Function `strtok` searches for the first character in `string` that's not a delimiting character (space).
- This begins the first token.
- The function then finds the next delimiting character in the string and *replaces it with a null (`'\0'`) character* to terminate the current token.
- Function `strtok` saves a pointer to the next character following the token in `string` and returns a pointer to the current token.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.8.7 Function strtok (Cont.)

- Subsequent `strtok` calls continue tokenizing `string`.
- These calls contain *`NULL` as their first argument*.
- The `NULL` argument indicates that the call to `strtok` should continue tokenizing from the location in `string` saved by the last call to `strtok`.
- If no tokens remain when `strtok` is called, `strtok` returns `NULL`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.8.7 Function strtok (Cont.)

- You can change the delimiter string in each new call to `strtok`.
- Figure 8.26 uses `strtok` to tokenize the string `"This is a sentence with 7 tokens"`.
- Each token is printed separately.
- Function `strtok` *modifies the input string* by placing `'\0'` at the end of each token; therefore, a *copy* of the string should be made if the string will be used again in the program after the calls to `strtok`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.9 Memory Functions of the String-Handling Library

- The string-handling library functions presented in this section manipulate, compare and search blocks of memory.
- The functions treat blocks of memory as character arrays and can manipulate any block of data.
- Figure 8.27 summarizes the memory functions of the string-handling library.
- In the function discussions, “object” refers to a block of data.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Function prototype	Function description
`void *memcpy(void *s1, const void *s2, size_t n);`	
	Copies `n` *bytes* from the object pointed to by `s2` into the object pointed to by `s1`. A pointer to the resulting object is returned.
`void *memmove(void *s1, const void *s2, size_t n);`	
	Copies `n` *bytes* from the object pointed to by `s2` into the object pointed to by `s1`. The copy is performed as if the bytes were first copied from the object pointed to by `s2` into a *temporary array* and then from the temporary array into the object pointed to by `s1`. A pointer to the resulting object is returned.
`int memcmp(const void *s1, const void *s2, size_t n);`	
	Compares the first `n` *bytes* of the objects pointed to by `s1` and `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`.
`void *memchr(const void *s, int c, size_t n);`	
	Locates the first occurrence of `c` (converted to `unsigned char`) in the first `n` bytes of the object pointed to by `s`. If `c` is found, a pointer to `c` in the object is returned. Otherwise, `NULL` is returned.

Fig. 8.27 | Memory functions of the string-handling library. (Part 1 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Function prototype	Function description
`void *memset(void *s, int c, size_t n);`	
	Copies c (converted to `unsigned char`) into the *first* n *bytes* of the object pointed to by `s`. A pointer to the result is returned.

Fig. 8.27 | Memory functions of the string-handling library. (Part 2 of 2.)

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.9 Memory Functions of the String-Handling Library (Cont.)

- The pointer parameters are declared `void *` so they can manipulate memory for any data type.
- In Chapter 7, we saw that a pointer to any data type can be assigned directly to a pointer of type `void *`, and a pointer of type `void *` can be assigned directly to a pointer to any data type.
- So, these functions can receive pointers to any data type.
- Because a `void *` pointer cannot be dereferenced, each function receives a size argument that specifies the number of characters (bytes) the function will process.
- For simplicity, the examples in this section manipulate character arrays (blocks of characters).

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.9.1 Function memcpy

- Function `memcpy` copies a specified number of characters from the object pointed to by its second argument into the object pointed to by its first argument.
- The function can receive a pointer to any type of object.
- The result of this function is *undefined* if the two objects overlap in memory (i.e., if they are parts of the same object)—in such cases, use `memmove`.
- Figure 8.31 uses `memcpy` to copy the string in array `s2` to array `s1`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Performance Tip 8.1

`memcpy` is more efficient than `strcpy` when you know the size of the string you are copying.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.28: fig08_28.c
// Using function memcpy
#include <stdio.h>
#include <string.h>

int main(void)
{
   char s1[17]; // create char array s1
   char s2[] = "Copy this string"; // initialize char array s2

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

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

Fig. 8.28 | Using function `memcpy`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.9.2 Function memmove

- Function `memmove`, like `memcpy`, *copies a specified number of bytes* from the object pointed to by its second argument into the object pointed to by its first argument.
- Copying is performed as if the bytes were copied from the second argument into a temporary character array, then copied from the temporary array into the first argument.
- This allows characters from one part of a string to be copied into another part of the *same* string.
- Figure 8.29 uses `memmove` to copy the last 10 bytes of array `x` into the first 10 bytes of array `x`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Common Programming Error 8.7

String-manipulation functions other than `memmove` *that copy characters have undefined results when copying takes place between parts of the* same *string.*

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.29: fig08_29.c
// Using function memmove
#include <stdio.h>
#include <string.h>

int main(void)
{
   char x[] = "Home Sweet Home"; // initialize char array x

   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: ",
      (char *) memmove(x, &x[5], 10));
}
```

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

Fig. 8.29 | Using function memmove.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.9.3 Function memcmp

- Function `memcmp` (Fig. 8.30) *compares the specified number of characters* of its first argument with the corresponding characters of its second argument.
- The function returns a value greater than 0 if the first argument is *greater than* the second, returns 0 if the arguments are equal and returns a value less than 0 if the first argument is less than the second.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.30: fig08_30.c
// Using function memcmp
#include <stdio.h>
#include <string.h>

int main(void)
{
   char s1[] = "ABCDEFG"; // initialize char array s1
   char s2[] = "ABCDXYZ"; // initialize char array s2

   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));
}
```

```
s1 = ABCDEFG
s2 = ABCDXYZ

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

Fig. 8.30 | Using function memcmp.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.9.4 Function `memchr`

- Function `memchr` searches for the *first occurrence of a byte*, represented as `unsigned char`, in the specified number of bytes of an object.
- If the byte is found, a pointer to the byte in the object is returned; otherwise, a `NULL` pointer is returned.
- Figure 8.31 searches for the character (byte) `'r'` in the string `"This is a string"`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.31: fig08_31.c
// Using function memchr
#include <stdio.h>
#include <string.h>

int main(void)
{
   const char *s = "This is a string"; // initialize char pointer

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

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

Fig. 8.31 | Using function memchr.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.9.5 Function `memset`

- Function `memset` copies the value of the byte in its second argument into the first *n* bytes of the object pointed to by its first argument, where *n* is specified by the third argument.
- Figure 8.32 uses `memset` to copy `'b'` into the first 7 bytes of `string1`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Performance Tip 8.2

Use `memset` *to set an array's elements to* `0` *rather than looping through them and assigning* `0` *to each element. For example, Fig. 6.3 could have initialized the five-element array* `n` *with* `memset(n, 0, 5);`*. Many hardware architectures have a block copy or clear instruction that the compiler can use to optimize* `memset` *for high-performance zeroing of memory.*

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.32: fig08_32.c
// Using function memset
#include <stdio.h>
#include <string.h>

int main(void)
{
   char string1[15] = "BBBBBBBBBBBBBB"; // initialize string1

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

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

Fig. 8.32 | Using function memset.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.10 Other Functions of the String-Handling Library

- The two remaining functions of the string-handling library are `strerror` and `strlen`.
- Figure 8.33 summarizes the `strerror` and `strlen` functions.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Function prototype	Function description
`char *strerror(int errornum);`	
	Maps `errornum` into a full text string in a compiler- and locale-specific manner (e.g. the message may appear in different spoken languages based on the computer's locale). A pointer to the string is returned. Error numbers are defined in `errno.h`.
`size_t strlen(const char *s);`	
	Determines the length of string `s`. The number of characters preceding the terminating null character is returned.

Fig. 8.33 | Other functions of the string-handling library.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.10.1 Function `strerror`

- Function `strerror` takes an error number and creates an error message string.
- A pointer to the string is returned.
- Figure 8.34 demonstrates `strerror`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.34: fig08_34.c
// Using function strerror
#include <stdio.h>
#include <string.h>

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

```
No such file or directory
```

Fig. 8.34 | Using function `strerror`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.10.2 Function `strlen`

- Function `strlen` takes a string as an argument and returns the number of characters in the string—the terminating null character is not included in the length.
- Figure 8.35 demonstrates function `strlen`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
// Fig. 8.35: fig08_35.c
// Using function strlen
#include <stdio.h>
#include <string.h>

int main(void)
{
   // initialize 3 char pointers
   const char *string1 = "abcdefghijklmnopqrstuvwxyz";
   const char *string2 = "four";
   const char *string3 = "Boston";

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

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

Fig. 8.35 | Using function `strlen`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.11 Secure C Programming

Secure String-Processing Functions

- In this chapter, we presented functions `sprintf`, `strcpy`, `strncpy`, `strcat`, `strncat`, `strtok`, `strlen`, `memcpy`, `memmove` and `memset`.
- More secure versions of these and many other string-processing and input/output functions are described by the C11 standard's optional Annex K.
- If your C compiler supports Annex K, you should use the secure versions of these functions.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.11 Secure C Programming (Cont.)

- Among other things, the more secure versions help prevent buffer overflows by requiring an additional parameter that specifies the number of elements in the target array and by ensuring that pointer arguments are non-`NULL`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.11 Secure C Programming (Cont.)

Reading Numeric Inputs and Input Validation

- It's important to validate the data that you input into a program.
- For example, when you ask the user to enter an integer in the range 1–100 then attempt to read that integer using `scanf`, there are several possible problems.
- The user could enter an integer that's outside the program's required range, an integer that's outside the allowed range for integers on that computer, a non-integer numeric value or a non-numeric value.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

8.11 Secure C Programming (Cont.)

- You can use various functions that you learned in this chapter to fully validate such input. For example, you could
  - use `fgets` to read the input as a line of text
  - convert the string to a number using `strtol` and ensure that the conversion was successful, then
  - ensure that the value is in range.
- For more information and techniques for converting input to numeric values, see CERT guideline INT05-C at `www.securecoding.cert.org`.

©2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.