C Characters and Strings

C How to Program, 8/e

8.2 Fundamentals of Strings and Characters (Cont.)

- Character: a value in ASCII code
  - `'z'` represents the integer value of `z`(122 in ASCII)
  - `'\n'` the integer value of newline (10 in ASCII)
  - '0' ⇒ 48, '9' ⇒ 57
  - 'A' ~ 'Z' ⇒ 65 ~ 90
  - 'a' ~ 'z' ⇒ 97 ~ 122
- A string is a series of characters treated as a single unit.
  - written in double quotation marks.
- A string in C is an array of characters ending in the null character (`'\0'`).

8.2 Fundamentals of Strings and Characters (Cont.)

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

  each initialize a variable to the string "blue".

- It is equivalent to
  - ```
    char color[] = { 'b', 'l', 'u', 'e', '\0' };
    ```

8.2 Fundamentals of Strings and Characters (Cont.)

- A string can be stored in an array using `scanf`.
- For example,
  - char `word[20];`
  - `scanf( "%s", word );`
    - `&` is not needed since `word` is the starting address of the array.
- `scanf` will read characters until a space, tab, newline or end-of-file indicator is encountered.

8.2 Fundamentals of Strings and Characters (Cont.)

- For a character array to be printed as a string, the array must contain a terminating null character.
  - printf(“%s”, word);

8.3 Character-Handling Library

- The character-handling library (`<ctype.h>`) includes several functions that perform useful tests and manipulations of character data.

- Figure 8.1 summarizes the functions of the character-handling library.

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.)

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.)

8.3 Character-Handling Library (Cont.)

- Function `isspace` determines if a character is one of the following white-space characters:
  - space (`' '`),
  - form feed (`'\f'`),
  - newline (`'\n'`),
  - carriage return (`'\r'`),
  - horizontal tab (`'\t'`)
  - vertical tab (`'\v'`).

8.3 Character-Handling Library (Cont.)

- Function `iscntrl` determines if 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'`)
  - newline (`'\n'`).

8.3 Character-Handling Library (Cont.)

- 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 if 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.

Example

```
1  // Fig. 8.2: fig08_02.c
2  // Using functions isdigit, isalpha, isalnum, and isxdigit
3  #include <stdio.h>
4  #include <ctype.h>
5
6  int main(void)
7  {
8     printf("%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
9        isdigit('8')   ? "8 is a " : "8 is not a ", "digit",
10       isdigit('#')   ? "# is a " : "# is not a ", "digit");
11
12    printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
13       "According to isalpha:",
14       isalpha('A')   ? "A is a " : "A is not a ", "letter",
15       isalpha('b')   ? "b is a " : "b is not a ", "letter",
16       isalpha('&')   ? "& is a " : "& is not a ", "letter",
17       isalpha('4')   ? "4 is a " : "4 is not a ", "letter");
18
```

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

```
19      printf("%s\n%s%s\n%s%s\n%s%s\n\n",
20         "According to isalnum:",
21         isalnum('A')   ? "A is a " : "A is not a ",
22         "digit or a letter",
23         isalnum('8')   ? "8 is a " : "8 is not a ",
24         "digit or a letter",
25         isalnum('#')   ? "# is a " : "# is not a ",
26         "digit or a letter");
27
28      printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n",
29         "According to isxdigit:",
30         isxdigit('F')   ? "F is a " : "F is not a ",
31         "hexadecimal digit",
32         isxdigit('J')   ? "J is a " : "J is not a ",
33         "hexadecimal digit",
34         isxdigit('7')   ? "7 is a " : "7 is not a ",
35         "hexadecimal digit",
36         isxdigit('$')   ? "$ is a " : "$ is not a ",
37         "hexadecimal digit",
38         isxdigit('f')   ? "f is a " : "f is not a ",
39         "hexadecimal digit");
40   }
```

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

```
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.)

8.3 Character-Handling Library (Cont.)

- 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.

8.4 String-Conversion Functions

- The string-conversion functions from the general utilities library (`<stdlib.h>`) convert strings of digits to integer and floating-point values.
- Figure 8.5 summarizes the string-conversion functions.

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.

```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`.

8.4 String-Conversion Functions (Cont.)

- Function `strtod` (Fig. 8.6) converts a sequence of characters representing a floating-point value to `double`.

- The pointer is assigned the location of the first character after the converted portion of the string.
  - `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`.

8.5 Standard Input/Output Library Functions

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.)

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.)

```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.)

```
22      // use puts to display sentence
23      puts("\nThe line entered was:");
24      puts(sentence);
25   }
```

```
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.)

```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.)

```
21  // recursively outputs characters in string in reverse order
22  void reverse(const char * const sPtr)
23  {
24     // if end of the string
25     if ('\0' == sPtr[0]) { // base case
26        return;
27     }
28     else { // if not end of the string
29        reverse(&sPtr[1]); // recursion step
30        putchar(sPtr[0]); // use putchar to display character
31     }
32  }
```

```
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.)

8.5 Standard Input/Output Library Functions (Cont.)

- Figure 8.15 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).

```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.)

```
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.)

8.6 String-Manipulation Functions of the String-Handling Library

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.

```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.)

```
19      // copy first 14 characters of x into z. Does not copy null
20      // character
21      strncpy(z, x, SIZE2 - 1);
22
23      z[SIZE2 - 1] = '\0'; // terminate string in z
24      printf("The string in array z is: %s\n", z);
25  }
```

```
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.)

```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.)

```
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.)

8.7 Comparison Functions of the String-Handling Library

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.

```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.)

```
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.)

8.8 Search Functions of the String-Handling Library

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.)

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.)

8.8 Search Functions of the String-Handling Library (Cont.)

- Function `strtok` (Fig. 8.29) 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).

```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.)

從前一次剩下的
地方往下找

```
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.)

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.)

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.)

```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`.

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

- 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 8.35 uses `memset` to copy `'b'` into the first `7` bytes of `string1`.

```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`.

8.10 Other Functions of the String-Handling Library

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.

8.10 Other Functions of the String-Handling Library (Cont.)

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

```
1  // Fig. 8.34: fig08_34.c
2  // Using function strerror
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     printf("%s\n", strerror(2));
9  }
```

```
No such file or directory
```

Fig. 8.34 | Using function `strerror`.

8.10 Other Functions of the String-Handling Library (Cont.)

- 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.

```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`.