Computer Programming I

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

C Characters and Strings

Objectives

- In this chapter, you'll learn
  - To use the functions of the character-handling library (**<ctype.h>**)
  - To use the string-conversion functions of the general utilities library (**<stdlib.h>**)
  - To use the string and character input/output functions of the standard input/output library (**<stdio.h>**)
  - To use the string-processing functions of the string handling library (**<string.h>**)

Introduction

- In this chapter, we introduce 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.

Fundamentals of Strings and Characters

- Characters are the fundamental building blocks of source programs.
- A character constant is an **`int`** value represented as a character in **single quotes**.
  - 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.

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 is appropriate to say that a string is a pointer—in fact, a pointer to the string's first character.
- A character array or a variable of type `char *` can be initialized with a string in a definition.

Fundamentals of Strings and Characters (Cont.)

- The definitions

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

- The first definition creates a 5-element array color containing the characters '**b**', '**l**', '**u**', '**e**' and '**\0**'.
  - **adjustable** (twice space occupation)
- The second definition creates pointer variable **colorPtr** that points to the string **"blue"** somewhere in memory.
  - **unadjustable**
- The preceding array definition could also have been written

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

Fundamentals of Strings and Characters (Cont.)

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.

Common Programming Error 8.2

Printing a "string" that does not contain a terminating null character is an error.

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 );
```

- It is possible that the user input could exceed 19 characters and that your program might crash!

Fundamentals of Strings and Characters (Cont.)

- For reading input lines of arbitrary length, there is a nonstandard—yet widely supported—function **`readline`**, usually included in **`<stdio.h>`**.
- For a character array to be printed as a string, the array must contain a terminating null character.

Fundamentals of Strings and Characters (Cont.)

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.

Character-Handling Library

- The character-handling library (**<ctype.h>**) includes several functions that perform useful tests and manipulations of character data.
- Each function receives a character—represented as an **int**—or **EOF** as an argument.
- **EOF** normally has the value **–1**, and some hardware architectures do not allow negative values to be stored in char variables, so the character-handling functions manipulate characters as integers.

Character-Handling Library (Cont.)

Prototype	Function description
`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 otherwise.
`int isalnum( int c );`	Returns a true value if `c` is a digit or a letter and 0 otherwise.
`int isxdigit( int c );`	Returns a true value if `c` is a hexadecimal digit character and 0 otherwise. (See Appendix C, Number Systems, 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 otherwise.
`int isupper( int c );`	Returns a true value if `c` is an uppercase letter and 0 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.
`int isspace( int c );`	Returns a true value if `c` is a white-space character—newline (`'\n'`), space (`' '`), form feed (`'\f'`), carriage return (`'\r'`), horizontal tab (`'\t'`) or vertical tab (`'\v'`)—and 0 otherwise.

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

Character-Handling Library (Cont.)

Prototype	Function description
`int iscntrl( int c );`	Returns a true value if c is a control character and 0 otherwise.
`int ispunct( int c );`	Returns a true value if c is a printing character other than a space, a digit, or a letter and returns 0 otherwise.
`int isprint( int c );`	Returns a true value if c is a printing character including a space (' ') and returns 0 otherwise.
`int isgraph( int c );`	Returns a true value if c is a printing character other than a space (' ') and returns 0 otherwise.

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

Character-Handling Library (Cont.)

- Example: fig08_02.c

```
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" );
```

Character-Handling Library (Cont.)

- Example: fig08_02.c

```
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" );
```

Character-Handling Library (Cont.)

- Example: fig08_02.c

```
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" );
```

Character-Handling Library (Cont.)

- Example: fig08_02.c

```
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" );
```

```
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
```

Character-Handling Library (Cont.)

- Conditional operator (**`? :`**)

```
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.

Character-Handling Library (Cont.)

- Example: fig08_03.c

```
30    printf( "%s%c\n%s%c\n%s%c\n%s%c\n",
31            "u converted to uppercase is ", toupper( 'u' ),
32            "7 converted to uppercase is ", toupper( '7' ),
33            "$ converted to uppercase is ", toupper( '$' ),
34            "L converted to lowercase is ", tolower( 'L' ) );
```

Character-Handling Library (Cont.)

- Example: fig08_03.c

```
30    printf( "%s%c\n%s%c\n%s%c\n%s%c\n",
31            "u converted to uppercase is ", toupper( 'u' ),
32            "7 converted to uppercase is ", toupper( '7' ),
33            "$ converted to uppercase is ", toupper( '$' ),
34            "L converted to lowercase is ", tolower( 'L' ) );
```

Character-Handling Library (Cont.)

- Example: fig08_03.c

```
30    printf( "%s%c\n%s%c\n%s%c\n%s%c\n",
31            "u converted to uppercase is ", toupper( 'u' ),
32            "7 converted to uppercase is ", toupper( '7' ),
33            "$ converted to uppercase is ", toupper( '$' ),
34            "L converted to lowercase is ", tolower( 'L' ) );
```

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

Character-Handling Library (Cont.)

- Example: fig08_04.c

```
17    printf( "%s\n%s%s%s\n%s%s\n\n", "According to iscntrl:",
18            "Newline", iscntrl( '\n' ) ? " is a " : " is not a ",
19            "control character", iscntrl( '$' ) ? "$ is a " :
20            "$ is not a ", "control character" );
21
22    printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
23            "According to ispunct:",
24            ispunct( ';' ) ? "; is a " : "; is not a ",
25            "punctuation character",
26            ispunct( 'Y' ) ? "Y is a " : "Y is not a ",
27            "punctuation character",
28            ispunct( '#' ) ? "# is a " : "# is not a ",
29            "punctuation character" );
30
31    printf( "%s\n%s%s\n%s%s%s\n\n", "According to isprint:",
32            isprint( '$' ) ? "$ is a " : "$ is not a ",
33            "printing character",
34            "Alert", isprint( '\a' ) ? " is a " : " is not a ",
35            "printing character" );
36
37    printf( "%s\n%s%s\n%s%s%s\n",  "According to isgraph:",
38            isgraph( 'Q' ) ? "Q is a " : "Q is not a ",
39            "printing character other than a space",
40            "Space", isgraph( ' ' ) ? " is a " : " is not a ",
41            "printing character other than a space" );
```

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`**') or newline ('**`\n`**').
- Function **`ispunct`** determines if a character is a printing character other than a space, a digit or a letter, such as **`$ # ( ) [ ]{ } ; : %`**

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.

String-Conversion Functions (Cont.)

Function prototype	Function description
`double atof( const char *nPtr );`	Converts the string `nPtr` to `double`.
`int atoi( const char *nPtr );`	Converts the string `nPtr` to `int`.
`long atol( const char *nPtr );`	Converts the string `nPtr` to `long int`.
`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.

String-Conversion Functions (Cont.)

- Example: fig08_06.c

```c
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
    double d; /* variable to hold converted string */

    d = atof( "99.0" );

    printf( "%s%.3f\n%s%.3f\n",
            "The string \"99.0\" converted to double is ", d,
            "The converted value divided by 2 is ", d / 2.0 );
    return 0; /* indicates successful termination */
} /* end main */
```

String-Conversion Functions (Cont.)

- Example: fig08_06.c

```c
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
    double d; /* variable to hold converted string */

    d = atof( "99.0" );

    printf( "%s%.3f\n%s%.3f\n",
            "The string \"99.0\" converted to double is ", d,
            "The converted value divided by 2 is ", d / 2.0 );
    return 0; /* indicates successful termination */
} /* end main */
```

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

String-Conversion Functions (Cont.)

- Function **`atof`** converts its argument—a string that represents a floating-point number—to a **`double`** value.
- If the converted value cannot be represented—for example, if the first character of the string is a letter—the behavior of function **`atof`** is undefined.

String-Conversion Functions (Cont.)

- Example: fig08_07.c

```
 8    int i; /* variable to hold converted string */
 9
10    i = atoi( "2593" );
11
12    printf( "%s%d\n%s%d\n",
13            "The string \"2593\" converted to int is ", i,
14            "The converted value minus 593 is ", i - 593 );
15    return 0; /* indicates successful termination */
```

String-Conversion Functions (Cont.)

- Example: fig08_07.c

```
 8    int i; /* variable to hold converted string */
 9
10    i = atoi( "2593" );
11
12    printf( "%s%d\n%s%d\n",
13            "The string \"2593\" converted to int is ", i,
14            "The converted value minus 593 is ", i - 593 );
15    return 0; /* indicates successful termination */
```

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

String-Conversion Functions (Cont.)

- Example: fig08_08.c

```
8     long l; /* variable to hold converted string */
9
10    l = atol( "1000000" );
11
12    printf( "%s%ld\n%s%ld\n",
13            "The string \"1000000\" converted to long int is ", l,
14            "The converted value divided by 2 is ", l / 2 );
15    return 0; /* indicates successful termination */
```

String-Conversion Functions (Cont.)

- Example: fig08_08.c

```
8     long l; /* variable to hold converted string */
9
10    l = atol( "1000000" );
11
12    printf( "%s%ld\n%s%ld\n",
13            "The string \"1000000\" converted to long int is ", l,
14            "The converted value divided by 2 is ", l / 2 );
15    return 0; /* indicates successful termination */
```

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

String-Conversion Functions (Cont.)

- Example: fig08_09.c

```
8     /* initialize string pointer */
9     const char *string = "51.2% are admitted"; /* initialize string */
10
11    double d; /* variable to hold converted sequence */
12    char *stringPtr; /* create char pointer */
13
14    d = strtod( string, &stringPtr );
15
16    printf( "The string \"%s\" is converted to the\n", string );
17    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"
```

String-Conversion Functions (Cont.)

- Example: fig08_09.c

```
8     /* initialize string pointer */
9     const char *string = "51.2% are admitted"; /* initialize string */
10
11    double d; /* variable to hold converted sequence */
12    char *stringPtr; /* create char pointer */
13
14    d = strtod( string, &stringPtr );
15
16    printf( "The string \"%s\" is converted to the\n", string );
17    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"
```

String-Conversion Functions (Cont.)

- Example: fig08_09.c

```
8     /* initialize string pointer */
9     const char *string = "51.2% are admitted"; /* initialize string */
10
11    double d; /* variable to hold converted sequence */
12    char *stringPtr; /* create char pointer */
13
14    d = strtod( string, &stringPtr );
15
16    printf( "The string \"%s\" is converted to the\n", string );
17    printf( "double value %.2f and the string \"%s\"\n", d, stringPtr );
```

strtod converts a string to double; it receives two arguments--a string (**char ***) and a pointer to a string (**char ****)

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

String-Conversion Functions (Cont.)

- Function **`strtod`**
  - The function receives two arguments—a string (**`char *`**) and a pointer to a string (**`char **`**).
  - The string contains the character sequence to be converted.
  - The pointer is assigned the location of the first character after the converted portion of the string.

String-Conversion Functions (Cont.)

- Example: fig08_10.c

```
 8    const char *string = "-1234567abc"; /* initialize string pointer */
 9    char *remainderPtr; /* create char pointer */
10    long x; /* variable to hold converted sequence */
11
12    x = strtol( string, &remainderPtr, 0 );
13
14    printf( "%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",
15            "The original string is ", string,
16            "The converted value is ", x,
17            "The remainder of the original string is ",
18            remainderPtr,
19            "The converted value plus 567 is ", x + 567 );
20    return 0; /* indicates successful termination */
```

String-Conversion Functions (Cont.)

- Example: fig08_10.c

```
 8    const char *string = "-1234567abc"; /* initialize string pointer */
 9    char *remainderPtr; /* create char pointer */
10    long x; /* variable to hold converted sequence */
11
12    x = strtol( string, &remainderPtr, 0 );
13
14    printf( "%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",
15            "The original string is ", string,
16            "The converted value is ", x,
17            "The remainder of the original string is ",
18            remainderPtr,
19            "The converted value plus 567 is ", x + 567 );
20    return 0; /* indicates successful termination */
```

String-Conversion Functions (Cont.)

- Example: fig08_10.c

```
 8    const char *string = "-1234567abc"; /* initialize string pointer */
 9    char *remainderPtr; /* create char pointer */
10    long x; /* variable to hold converted sequence */
11
12    x = strtol( string, &remainderPtr, 0 );
13
14    printf( "%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",
15            "The original string is ", string,
16            "The converted value is ", x,
17            "The remainder of the original string is ",
18            remainderPtr,
19            "The converted value plus 567 is ", x + 567 );
20    return 0; /* indicates successful termination */
```

```
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
```

String-Conversion Functions (Cont.)

- **`strtol`** function

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

  - 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.

String-Conversion Functions (Cont.)

- Example: fig08_10.c

```
8     const char *string = "1234567abc"; /* initialize string pointer */
9     unsigned long x; /* variable to hold converted sequence */
10    char *remainderPtr; /* create char pointer */
11
12    x = strtoul( string, &remainderPtr, 0 );
13
14    printf( "%s\"%s\"\n%s%lu\n%s\"%s\"\n%s%lu\n",
15            "The original string is ", string,
16            "The converted value is ", x,
17            "The remainder of the original string is ",
18            remainderPtr,
19            "The converted value minus 567 is ", x - 567 );
```

String-Conversion Functions (Cont.)

- Example: fig08_10.c

```
 8    const char *string = "1234567abc"; /* initialize string pointer */
 9    unsigned long x; /* variable to hold converted sequence */
10    char *remainderPtr; /* create char pointer */
11
12    x = strtoul( string, &remainderPtr, 0 );
13
14    printf( "%s\"%s\"\n%s%lu\n%s\"%s\"\n%s%lu\n",
15            "The original string is ", string,
16            "The converted value is ", x,
17            "The remainder of the original string is ",
18            remainderPtr,
19            "The converted value minus 567 is ", x - 567 );
```

String-Conversion Functions (Cont.)

- Example: fig08_10.c

```
 8    const char *string = "1234567abc"; /* initialize string pointer */
 9    unsigned long x; /* variable to hold converted sequence */
10    char *remainderPtr; /* create char pointer */
11
12    x = strtoul( string, &remainderPtr, 0 );
13
14    printf( "%s\"%s\"\n%s%lu\n%s\"%s\"\n%s%lu\n",
15            "The original string is ", string,
16            "The converted value is ", x,
17            "The remainder of the original string is ",
18            remainderPtr,
19            "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
```

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.

Standard Input/Output Library Functions (Cont.)

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`.
`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.
`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.

Standard Input/Output Library Functions (Cont.)

Function prototype	Function description
`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.

Standard Input/Output Library Functions (Cont.)

- Example: fig08_13.c

```
 8    char sentence[ 80 ]; /* create char array */
 9
10    printf( "Enter a line of text:\n" );
11
12    /* use fgets to read line of text */
13    fgets( sentence, 80, stdin );
14
15    printf( "\nThe line printed backward is:\n" );
16    reverse( sentence );
17    printf("\n");
18    return 0; /* indicates successful termination */
```

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

Standard Input/Output Library Functions (Cont.)

- Example: fig08_13.c

```
 8    char sentence[ 80 ]; /* create char array */
 9
10    printf( "Enter a line of text:\n" );
11
12    /* use fgets to read line of text */
13    fgets( sentence, 80, stdin );
14
15    printf( "\nThe line printed backward is:\n" );
16    reverse( sentence );
17    printf("\n");
18    return 0; /* indicates successful termination */
```

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

Standard Input/Output Library Functions (Cont.)

- Example: fig08_13.c

```
 8    char sentence[ 80 ]; /* create char array */
 9
10    printf( "Enter a line of text:\n" );
11
12    /* use fgets to read line of text */
13    fgets( sentence, 80, stdin );
14
15    printf( "\nThe line printed backward is:\n" );
16    reverse( sentence );
17    printf("\n");
18    return 0; /* indicates successful termination */
```

read a line of text from the standard input; it's a safer way to read input because of the second argument

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

Standard Input/Output Library Functions (Cont.)

- Example: fig08_13.c

```
 8    char sentence[ 80 ]; /* create char array */
 9
10    printf( "Enter a line of text:\n" );
11
12    /* use fgets to read line of text */
13    fgets( sentence, 80, stdin );
14
15    printf( "\nThe line printed backward is:\n" );
16    reverse( sentence );
17    printf("\n");
18    return 0; /* indicates successful termination */
```

read a line of text from the standard input; it's a safer way to read input because of the second argument

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

Standard Input/Output Library Functions (Cont.)

- Example: fig08_13.c

```
 8    char sentence[ 80 ]; /* create char array */
 9
10    printf( "Enter a line of text:\n" );
11
12    /* use fgets to read line of text */
13    fgets( sentence, 80, stdin );
14
15    printf( "\nThe line printed backward is:\n" );
16    reverse( sentence );
17    printf("\n");
18    return 0; /* indicates successful termination */
```

read a line of text from the standard input; it's a safer way to read input because of the second argument

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

putchar() prints its character argument

Standard Input/Output Library Functions (Cont.)

- Example: fig08_14.c

```
6     char c; /* variable to hold character input by user */
7     char sentence[ 80 ]; /* create char array */
8     int i = 0; /* initialize counter i */
9
10    /* prompt user to enter line of text */
11    puts( "Enter a line of text:" );
12
13    /* use getchar to read each character */
14    while ( ( c = getchar() ) != '\n') {
15        sentence[ i++ ] = c;
16    } /* end while */
17
18    sentence[ i ] = '\0'; /* terminate string */
19
20    /* use puts to display sentence */
21    puts( "\nThe line entered was:" );
22    puts( sentence );
```

Standard Input/Output Library Functions (Cont.)

- Example: fig08_14.c

```
6     char c; /* variable to hold character input by user */
7     char sentence[ 80 ]; /* create char array */
8     int i = 0; /* initialize counter i */
9
10    /* prompt user to enter line of text */
11    puts( "Enter a line of text:" );
12
13    /* use getchar to read each character */
14    while ( ( c = getchar() ) != '\n') {
15        sentence[ i++ ] = c;
16    } /* end while */
17
18    sentence[ i ] = '\0'; /* terminate string */
19
20    /* use puts to display sentence */
21    puts( "\nThe line entered was:" );
22    puts( sentence );
```

Standard Input/Output Library Functions (Cont.)

- Example: fig08_14.c

```
6     char c; /* variable to hold character input by user */
7     char sentence[ 80 ]; /* create char array */
8     int i = 0; /* initialize counter i */
9
10    /* prompt user to enter line of text */
11    puts( "Enter a line of text:" );
12
13    /* use getchar to read each character */
14    while ( ( c = getchar() ) != '\n') {
15        sentence[ i++ ] = c;
16    } /* end while */
17
18    sentence[ i ] = '\0'; /* terminate string */
19
20    /* use puts to display sentence */
21    puts( "\nThe line entered was:" );
22    puts( sentence );
```

prints the string followed by a newline character

Standard Input/Output Library Functions (Cont.)

- Example: fig08_14.c

```
6     char c; /* variable to hold character input by user */
7     char sentence[ 80 ]; /* create char array */
8     int i = 0; /* initialize counter i */
9
10    /* prompt user to enter line of text */
11    puts( "Enter a line of text:" );
12
13    /* use getchar to read each character */
14    while ( ( c = getchar() ) != '\n') {
15        sentence[ i++ ] = c;
16    } /* end while */
17
18    sentence[ i ] = '\0'; /* terminate string */
19
20    /* use puts to display sentence */
21    puts( "\nThe line entered was:" );
22    puts( sentence );
```

prints the string followed by a newline character

Standard Input/Output Library Functions (Cont.)

- Example: fig08_14.c

```
6     char c; /* variable to hold character input by user */
7     char sentence[ 80 ]; /* create char array */
8     int i = 0; /* initialize counter i */
9
10    /* prompt user to enter line of text */
11    puts( "Enter a line of text:" );
12
13    /* use getchar to read each character */
14    while ( ( c = getchar() ) != '\n') {
15        sentence[ i++ ] = c;
16    } /* end while */
17
18    sentence[ i ] = '\0'; /* terminate string */
19
20    /* use puts to display sentence */
21    puts( "\nThe line entered was:" );
22    puts( sentence );
```

prints the string followed by a newline character

read a sequence of characters into the **sentence** array

Standard Input/Output Library Functions (Cont.)

- Example: fig08_14.c

```
6     char c; /* variable to hold character input by user */
7     char sentence[ 80 ]; /* create char array */
8     int i = 0; /* initialize counter i */
9
10    /* prompt user to enter line of text */
11    puts( "Enter a line of text:" );
12
13    /* use getchar to read each character */
14    while ( ( c = getchar() ) != '\n') {
15        sentence[ i++ ] = c;
16    } /* end while */
17
18    sentence[ i ] = '\0'; /* terminate string */
19
20    /* use puts to display sentence */
21    puts( "\nThe line entered was:" );
22    puts( sentence );
```

prints the string followed by a newline character

read a sequence of characters into the **sentence** array

Standard Input/Output Library Functions (Cont.)

- Example: fig08_14.c

```
6     char c; /* variable to hold character input by user */
7     char sentence[ 80 ]; /* create char array */
8     int i = 0; /* initialize counter i */
9
10    /* prompt user to enter line of text */
11    puts( "Enter a line of text:" );
12
13    /* use getchar to read each character */
14    while ( ( c = getchar() ) != '\n') {
15        sentence[ i++ ] = c;
16    } /* end while */
17
18    sentence[ i ] = '\0'; /* terminate string */
19
20    /* use puts to display sentence */
21    puts( "\nThe line entered was:" );
22    puts( sentence );
```

prints the string followed by a newline character

read a sequence of characters into the **sentence** array

prints these strings

Standard Input/Output Library Functions (Cont.)

- Example: fig08_15.c

```
6     char s[ 80 ]; /* create char array */
7     int x; /* x value to be input */
8     double y; /* y value to be input */
9
10    printf( "Enter an integer and a double:\n" );
11    scanf( "%d%lf", &x, &y );
12
13    sprintf( s, "integer:%6d\ndouble:%8.2f", x, y );
14
15    printf( "%s\n%s\n",
16            "The formatted output stored in array s is:", s );
17    return 0; /* indicates successful termination */
```

Standard Input/Output Library Functions (Cont.)

- Example: fig08_15.c

```
 6     char s[ 80 ]; /* create char array */
 7     int x; /* x value to be input */
 8     double y; /* y value to be input */
 9
10     printf( "Enter an integer and a double:\n" );
11     scanf( "%d%lf", &x, &y );
12
13     sprintf( s, "integer:%6d\ndouble:%8.2f", x, y );
14
15     printf( "%s\n%s\n",
16             "The formatted output stored in array s is:", s );
17     return 0; /* indicates successful termination */
```

Standard Input/Output Library Functions (Cont.)

- Example: fig08_15.c

```
6     char s[ 80 ]; /* create char array */
7     int x; /* x value to be input */
8     double y; /* y value to be input */
9
10    printf( "Enter an integer and a double:\n" );
11    scanf( "%d%lf", &x, &y );
12
13    sprintf( s, "integer:%6d\ndouble:%8.2f", x, y );
14
15    printf( "%s\n%s\n",
16            "The formatted output stored in array s is:", s );
17    return 0; /* indicates successful termination */
```

sprintf() prints formatted data into array **s** -- an array of characters

Standard Input/Output Library Functions (Cont.)

- Example: fig08_15.c

```
6     char s[ 80 ]; /* create char array */
7     int x; /* x value to be input */
8     double y; /* y value to be input */
9
10    printf( "Enter an integer and a double:\n" );
11    scanf( "%d%lf", &x, &y );
12
13    sprintf( s, "integer:%6d\ndouble:%8.2f", x, y );
14
15    printf( "%s\n%s\n",
16            "The formatted output stored in array s is:", s );
17    return 0; /* indicates successful termination */
```

sprintf() prints formatted data into array **s** -- an array of characters

```
Enter an integer and a double:
10 3.14
The formatted output stored in array s is:
integer:    10
double:    3.14
```

Standard Input/Output Library Functions (Cont.)

- Example: fig08_16.c

```
6     char s[] = "31298 87.375"; /* initialize array s */
7     int x; /* x value to be input */
8     double y; /* y value to be input */
9
10    sscanf( s, "%d%lf", &x, &y );
11    printf( "%s\n%s%6d\n%s%8.3f\n",
12            "The values stored in character array s are:",
13            "integer:", x, "double:", y );
14    return 0; /* indicates successful termination */
```

Standard Input/Output Library Functions (Cont.)

- Example: fig08_16.c

```
6     char s[] = "31298 87.375"; /* initialize array s */
7     int x; /* x value to be input */
8     double y; /* y value to be input */
9
10    sscanf( s, "%d%lf", &x, &y );
11    printf( "%s\n%s%6d\n%s%8.3f\n",
12            "The values stored in character array s are:",
13            "integer:", x, "double:", y );
14    return 0; /* indicates successful termination */
```

Standard Input/Output Library Functions (Cont.)

- Example: fig08_16.c

```
6     char s[] = "31298 87.375"; /* initialize array s */
7     int x; /* x value to be input */
8     double y; /* y value to be input */
9
10    sscanf( s, "%d%lf", &x, &y );
11    printf( "%s\n%s%6d\n%s%8.3f\n",
12            "The values stored in character array s are:",
13            "integer:", x, "double:", y );
14    return 0; /* indicates successful termination */
```

sscanf() reads formatted data from character array **s**

Standard Input/Output Library Functions (Cont.)

- Example: fig08_16.c

```
6     char s[] = "31298 87.375"; /* initialize array s */
7     int x; /* x value to be input */
8     double y; /* y value to be input */
9
10    sscanf( s, "%d%lf", &x, &y );
11    printf( "%s\n%s%6d\n%s%8.3f\n",
12            "The values stored in character array s are:",
13            "integer:", x, "double:", y );
14    return 0; /* indicates successful termination */
```

sscanf() reads formatted data from character array **s**

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