BAKI ALİ NEFT MƏKTƏBİ
BAKU HIGHER OIL SCHOOL

Programming and Computer Applications-1

Characters and Strings

Instructor : PhD, Associate Professor Leyla Muradkhanli

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.

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.

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.
- 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.

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.
    ```

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.

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`.
- `scanf` will read characters until a space, tab, newline or end-of-file indicator is encountered.
- So, it is possible that the user input could exceed 19 characters and that your program might crash!

Fundamentals of Strings and Characters (Cont.)

- For this reason, use the conversion specifier `%19s` so that `scanf` reads up to 19 characters and saves the last character for the terminating null character.
- This prevents `scanf` from writing characters into memory beyond the end of `s`.
- (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.

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

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.)

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.)

- 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`).

```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 `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      return 0; /* indicates successful termination */
41   } /* end main */
```

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

Character-Handling Library (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.

Character-Handling Library (Cont.)

- 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.

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.

```c
/* Fig. 8.3: fig08_03.c
   Using functions islower, isupper, tolower, 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.)

```
19      printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
20         "According to isupper:",
21         isupper( 'D' ) ? "D is an " : "D is not an ",
22         "uppercase letter",
23         isupper( 'd' ) ? "d is an " : "d is not an ",
24         "uppercase letter",
25         isupper( '8' ) ? "8 is an " : "8 is not an ",
26         "uppercase letter",
27         isupper( '$' ) ? "$ is an " : "$ is not an ",
28         "uppercase letter" );
29
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' ) );
35      return 0; /* indicates successful termination */
36   } /* end main */
```

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

```
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.)

Character-Handling Library (Cont.)

- Figure 8.4 demonstrates functions `isspace`, `iscntrl`, `ispunct`, `isprint` and `isgraph`.
- 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'`) or vertical tab (`'\v'`).

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 `$`, `#`, `(`, `)`, `[`, `]`, `{`, `}`, `;`, `:` 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.

```c
/* Fig. 8.4: fig08_04.c
   Using functions isspace, iscntrl, ispunct, isprint, 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 `isspace`, `iscntrl`, `ispunct`, `isprint` and `isgraph`. (Part 1 of 3.)

```
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" );
42      return 0; /* indicates successful termination */
43  } /* end main */
```

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

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

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.
- 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.

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.)

- Function `atof` (Fig. 8.6) converts its argument—a string that represents a floating-point number—to a `double` value.
- The function returns the `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.

```c
/* Fig. 8.6: fig08_06.c
   Using atof */
#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
```

Fig. 8.6 | Using atof.

String-Conversion Functions (Cont.)

- Function `atoi` (Fig. 8.7) converts its argument—a string of digits that represents an integer—to an `int` value.
- The function returns the `int` value.
- If the converted value cannot be represented, the behavior of function `atoi` is undefined.

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

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

   i = atoi( "2593" );

   printf( "%s%d\n%s%d\n",
      "The string \"2593\" converted to int is ", i,
      "The converted value minus 593 is ", i - 593 );
   return 0; /* indicates successful termination */
} /* end main */
```

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

Fig. 8.7 | Using `atoi`.

String-Conversion Functions (Cont.)

- Function `atol` (Fig. 8.8) converts its argument—a string of digits representing a long integer—to a `long` value.
- The function returns the `long` value.
- If the converted value cannot be represented, the behavior of function `atol` is undefined.
- If `int` and `long` are both stored in 4 bytes, function `atoi` and function `atol` work identically.

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

int main( void )
{
   long l; /* variable to hold converted string */

   l = atol( "1000000" );

   printf( "%s%ld\n%s%ld\n",
      "The string \"1000000\" converted to long int is ", l,
      "The converted value divided by 2 is ", l / 2 );
   return 0; /* indicates successful termination */
} /* end main */
```

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

Fig. 8.8 | Using `atol`.

String-Conversion Functions (Cont.)

- Function `strtod` (Fig. 8.9) converts a sequence of characters representing a floating-point value to `double`.
- 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. Line 14
    - `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`.

```c
/* Fig. 8.9: fig08_09.c
   Using strtod */
#include <stdio.h>
#include <stdlib.h>

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

   double d; /* variable to hold converted sequence */
   char *stringPtr; /* create char pointer */

   d = strtod( string, &stringPtr );

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

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

Fig. 8.9 | Using `strtod`.

String-Conversion Functions (Cont.)

- Function `strtol` (Fig. 8.10) converts to `long` a sequence of characters representing an integer.
- The function receives three arguments—a string (`char *`), a pointer to a string and an integer.
- 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.
- The integer specifies the base of the value being converted.

String-Conversion Functions (Cont.)

- Line 13
  - `x = strtol( string, &remainderPtr, 0 );`

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

- 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.
- For example, hexadecimal values can consist of the digits 0–9 and the characters A–F.
- A base-36 integer can consist of the digits 0–9 and the characters A–Z.

```c
/* Fig. 8.10: fig08_10.c
   Using strtol */
#include <stdio.h>
#include <stdlib.h>

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

   char *remainderPtr; /* create char pointer */
   long x; /* variable to hold converted sequence */

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

   printf( "%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",
      "The original string is ", string,
      "The converted value is ", x,
      "The remainder of the original string is ",
      remainderPtr,
      "The converted value plus 567 is ", x + 567 );
   return 0; /* indicates successful termination */
} /* end main */
```

Fig. 8.10 | Using `strtol`. (Part 1 of 2.)

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

Fig. 8.10 | Using `strtol`. (Part 2 of 2.)

String-Conversion Functions (Cont.)

- Function `strtoul` (Fig. 8.11) converts to `unsigned long` characters representing an `unsigned long` integer.
- The function works identically to function `strtol`.
- The statement
  - `x = strtoul( string, &remainderPtr, 0 );`

  in Fig. 8.11 indicates that `x` is assigned the `unsigned long` value converted from `string`.
- 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.

```c
/* Fig. 8.11: fig08_11.c
   Using strtoul */
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
   const char *string = "1234567abc"; /* initialize string pointer */
   unsigned long x; /* variable to hold converted sequence */
   char *remainderPtr; /* create char pointer */

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

   printf( "%s\"%s\"\n%s%lu\n%s\"%s\"\n%s%lu\n",
      "The original string is ", string,
      "The converted value is ", x,
      "The remainder of the original string is ",
      remainderPtr,
      "The converted value minus 567 is ", x - 567 );
   return 0; /* indicates successful termination */
} /* end main */
```

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

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

Fig. 8.11 | Using `strtoul`. (Part 2 of 2.)

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.

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.

Fig. 8.12 | Standard input/output library character and string functions. (Part 1 of 2.)

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.

Fig. 8.12 | Standard input/output library character and string functions. (Part 2 of 2.)

Standard Input/Output Library Functions (Cont.)

- Figure 8.13 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.

Standard Input/Output Library Functions (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.

Standard Input/Output Library Functions (Cont.)

- Otherwise, `reverse` is called again with the address of the subarray beginning at element `s[1]`, and character `s[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.

```c
/* Fig. 8.13: fig08_13.c
   Using gets and putchar */
#include <stdio.h>

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

int main( void )
{
   char sentence[ 80 ]; /* create char array */

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

   /* use fgets to read line of text */
   fgets( sentence, 80, stdin );

   printf( "\nThe line printed backward is:\n" );
   reverse( sentence );
   return 0; /* indicates successful termination */
} /* end main */

```

Fig. 8.13 | Using `fgets` and `putchar`. (Part 1 of 3.)

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

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

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

Fig. 8.13 | Using `fgets` and `putchar`. (Part 2 of 3.)

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

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

Fig. 8.13 | Using `fgets` and `putchar`. (Part 3 of 3.)

Standard Input/Output Library Functions (Cont.)

- Figure 8.14 uses functions `getchar` and `puts` to read characters from the standard input into character array `sentence` and print the array of 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 (`char *`) as an argument and prints the string followed by a newline character.
- The program stops inputting characters when `getchar` reads the newline character entered by the user to end the line.
- A null character is appended to array `sentence` (line 19) so that the array may be treated as a string.
- Then, function `puts` prints the string contained in `sentence`.

```c
/* Fig. 8.14: fig08_14.c
   Using getchar and puts */
#include <stdio.h>

int main( void )
{
   char c; /* variable to hold character input by user */
   char sentence[ 80 ]; /* 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 ( ( c = getchar() ) != '\n') {
      sentence[ i++ ] = c;
   } /* end while */

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

   /* use puts to display sentence */
   puts( "\nThe line entered was:" );
   puts( sentence );
   return 0; /* indicates successful termination */
} /* end main */
```

Fig. 8.14 | Using getchar and puts. (Part 1 of 2.)

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

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

Fig. 8.14 | Using `getchar` and `puts`. (Part 2 of 2.)

Standard Input/Output Library Functions (Cont.)

- Figure 8.15 uses function `sprintf` to print formatted data into array `s`—an array of characters.
- 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`.

```c
/* Fig. 8.15: fig08_15.c
   Using sprintf */
#include <stdio.h>

int main( void )
{
   char s[ 80 ]; /* create char array */
   int x; /* x value to be input */
   double y; /* y value to be input */

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

   sprintf( s, "integer:%6d\ndouble:%8.2f", x, y );

   printf( "%s\n%s\n",
           "The formatted output stored in array s is:", s );
   return 0; /* indicates successful termination */
} /* end main */
```

Fig. 8.15 | Using `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. 8.15 | Using `sprintf`. (Part 2 of 2.)

Standard Input/Output Library Functions (Cont.)

- Figure 8.16 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`.

```c
/* Fig. 8.16: fig08_16.c
   Using 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 );
   return 0; /* indicates successful termination */
} /* end main */
```

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

Fig. 8.16 | Using `sscanf`.