Lecture#4

Fundamentals of Strings and Characters

Character-handling library (<ctype.h>)

<stdio.h> for manipulating character and string data

String-handling library (<string.h>)

CENG 102- Algorithms and Programming II,

2024-2025, Spring

Contains materials from:
P. Deitel, H. Deitel, ''C How to Program with an Introduction to C++'', 8th edition, Pearson

Lecture 5.1

Fundamentals of Strings and Characters

8.2 Fundamentals of Strings and Characters

- Every program is composed of a sequence of characters that 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.
- For example, `'z'` represents the integer value of `z`, and `'\n'` the integer value of newline (122 and 10 in ASCII, respectively).

© 2016 Pearson Education, Ltd. All rights reserved.

8.2 Fundamentals of Strings and Characters (Cont.)

- 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**.
  - Single quote: '
  - Double quote: "

© 2016 Pearson Education, Ltd. All rights reserved.

8.2 Fundamentals of Strings and Characters (Cont.)

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

© 2016 Pearson Education, Ltd. All rights reserved.

8.2 Fundamentals of Strings and Characters (Cont.)

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

Portability Tip 8.1

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

8.2 Fundamentals of Strings and Characters (Cont.)

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

© 2016 Pearson Education, Ltd. All rights reserved.

Common Programming Error 8.1

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

© 2016 Pearson Education, Ltd. All rights reserved.

Common Programming Error 8.2

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

© 2016 Pearson Education, Ltd. All rights reserved.

Error-Prevention Tip 8.1

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

© 2016 Pearson Education, Ltd. All rights reserved.

8.2 Fundamentals of Strings and Characters (Cont.)

- A string can be stored in an array using `scanf`.
- For example, the following statement stores a string in character array `word[10]`:
    - ```
      char word[10];
      scanf("%s", word);
      ```
- The string entered by the user is stored in `word`.
- Variable `word` is an array, which is, of course, a pointer, so the & is not needed with argument `word`.

© 2016 Pearson Education, Ltd. All rights reserved.
      ```

8.2 Fundamentals of Strings and Characters (Cont.)

- `scanf` will read characters **until a space, tab, newline or end-of-file** indicator is encountered.
- So, it's possible that, without the field width `9` in the conversion specifier `%9s`, the user input could exceed 9 characters and that your program might crash!
- For this reason, you should ***always* use a field width** when using `scanf` to read into a `char` array.
- The field width `9` in the preceding statement ensures that `scanf` reads a *maximum* of 9 characters and saves the last character for the string's terminating null character.

© 2016 Pearson Education, Ltd. All rights reserved.

8.2 Fundamentals of Strings and Characters (Cont.)

- This prevents `scanf` from writing characters into memory beyond the end of `s`.
- For a character array to be printed properly as a string, the array must contain a terminating null character.

© 2016 Pearson Education, Ltd. All rights reserved.

Common Programming Error 8.4

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

© 2016 Pearson Education, Ltd. All rights reserved.

Lecture 5.2

Character-handling library (<ctype.h>)

8.3 Character-Handling Library

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

int main(void)
{
   printf("%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
      isdigit('8')   ? "8 is a " : "8 is not a ", "digit",
      isdigit('#')   ? "# is a " : "# is not a ", "digit");

   printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
      "According to isalpha:",
      isalpha('A')   ? "A is a " : "A is not a ", "letter",
      isalpha('b')   ? "b is a " : "b is not a ", "letter",
      isalpha('&')   ? "& is a " : "& is not a ", "letter",
      isalpha('4')   ? "4 is a " : "4 is not a ", "letter");

```

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

© 2016 Pearson Education, Ltd. All rights reserved.

```
    printf("%s\n%s%s\n%s%s\n%s%s\n\n",
       "According to isalnum:",
       isalnum('A')  ? "A is a " : "A is not a ",
       "digit or a letter",
       isalnum('8')  ? "8 is a " : "8 is not a ",
       "digit or a letter",
       isalnum('#')  ? "# is a " : "# is not a ",
       "digit or a letter");

    printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n",
       "According to isxdigit:",
       isxdigit('F')  ? "F is a " : "F is not a ",
       "hexadecimal digit",
       isxdigit('J')  ? "J is a " : "J is not a ",
       "hexadecimal digit",
       isxdigit('7')  ? "7 is a " : "7 is not a ",
       "hexadecimal digit",
       isxdigit('$')  ? "$ is a " : "$ is not a ",
       "hexadecimal digit",
       isxdigit('f')  ? "f is a " : "f is not a ",
       "hexadecimal digit");
}
```

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

- Figure 8.3 demonstrates functions `islower`, `isupper`, `tolower` and `toupper`.

- Function `islower` determines whether its argument is a lowercase letter (`a`–`z`).
- Function `isupper` determines whether its argument is an uppercase letter (`A`–`Z`).

© 2016 Pearson Education, Ltd. All rights reserved.

8.3 Character-Handling Library (Cont.)

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

int main(void)
{
   printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
      "According to islower:",
      islower('p')  ? "p is a " : "p is not a ",
      "lowercase letter",
      islower('P')  ? "P is a " : "P is not a ",
      "lowercase letter",
      islower('5')  ? "5 is a " : "5 is not a ",
      "lowercase letter",
      islower('!')  ? "! is a " : "! is not a ",
      "lowercase letter");

```

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

© 2016 Pearson Education, Ltd. All rights reserved.

```c
    printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
       "According to isupper:",
       isupper('D')  ? "D is an " : "D is not an ",
       "uppercase letter",
       isupper('d')  ? "d is an " : "d is not an ",
       "uppercase letter",
       isupper('8')  ? "8 is an " : "8 is not an ",
       "uppercase letter",
       isupper('$')  ? "$ is an " : "$ is not an ",
       "uppercase letter");

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

- Figure 8.4 demonstrates functions `isspace`, `iscntrl`, `ispunct`, `isprint` and `isgraph`.

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

int main(void)
{
   printf("%s\n%s%s%s\n%s%s%s\n%s%s\n\n",
       "According to isspace:",
       "Newline", isspace('\n')  ? " is a " : " is not a ",
       "whitespace character", "Horizontal tab",
       isspace('\t')  ? " is a " : " is not a ",
       "whitespace character",
       isspace('%')  ? "% is a " : "% is not a ",
       "whitespace character");

   printf("%s\n%s%s%s\n%s%s\n\n", "According to iscntrl:",
       "Newline", iscntrl('\n')  ? " is a " : " is not a ",
       "control character", iscntrl('$')  ? "$ is a " :
       "$ is not a ", "control character");

```

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

© 2016 Pearson Education, Ltd. All rights reserved.

```
    printf("%s\n%s%s\n%s%s\n%s%s\n\n",
        "According to ispunct:",
        ispunct(';')  ? "; is a " : "; is not a ",
        "punctuation character",
        ispunct('Y')  ? "Y is a " : "Y is not a ",
        "punctuation character",
        ispunct('#')  ? "# is a " : "# is not a ",
        "punctuation character");

    printf("%s\n%s%s\n%s%s%s\n\n", "According to isprint:",
        isprint('$')  ? "$ is a " : "$ is not a ",
        "printing character",
        "Alert", isprint('\a')  ? " is a " : " is not a ",
        "printing character");

    printf("%s\n%s%s\n%s%s%s\n",  "According to isgraph:",
        isgraph('Q')  ? "Q is a " : "Q is not a ",
        "printing character other than a space",
        "Space", isgraph(' ')  ? " is a " : " is not a ",
        "printing character other than a space");
}
```

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

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

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

Lecture 5.3

<stdio.h> for manipulating character and string data

8.5 Standard Input/Output Library Functions

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

```c
char result[100];
int age = 10;
char name[] = "John";
sprintf(result, "My name is %s and I am %d years old.", name, age);
printf("%s\n", result);  // "My name is John and I am 10 years old."
```

```c
char str[] = "John 10";
char name[20];
int age;
sscanf(str, "%s %d", name, &age);
printf("Name: %s, Age: %d\n", name, age);  // "Name: John, Age: 10"
```

© 2016 Pearson Education, Ltd. All rights reserved.

8.5.1 Functions `fgets` and `putchar`

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

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

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

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

```

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

8.5.2 Function getchar

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

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

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

```

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

8.5.3 Function `sprintf`

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

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

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

}
```

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

8.5.4 Function `sscanf`

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

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

Fig. 8.13 | Using function `sscanf`.

© 2016 Pearson Education, Ltd. All rights reserved.

Lecture 5.4

String-handling library (<string.h>)

8.6 String-Manipulation Functions of the String-Handling Library

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

- Function `strcpy` copies its second argument (a string) into its first argument—a character array that *you must ensure is large enough* to store the string and its terminating null character, which is also copied.
- Function `strncpy` is equivalent to `strcpy`, except that `strncpy` specifies the number of characters to be copied from the string into the array.
- *Function `strncpy` does not necessarily copy the terminating null character of its second argument.*
- *This occurs only if the number of characters to be copied (n) is at least one more than the length of the string.*

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

int main()
{
    char str1[20] = "Hello";
    char str2[20];

    strcpy(str2, str1);

    printf("str1: %s\n", str1);
    printf("str2: %s\n", str2);

    return 0;
}
```

Output

```
str1: Hello
str2: Hello
```

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

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

int main()
{
    char str1[] = "Hello World!";
    char str2[] = "GfG";
    char str3[40];
    char str4[40];
    char str5[] = "GeeksForGeeks";

    strcpy(str2, str1);
    strcpy(str3, "Copy successful");
    strcpy(str4, str5);
    printf("str1: %s\nstr2: %s\nstr3: %s\nstr4:%s\n", str1,
           str2, str3, str4);
    return 0;
}
```

Output

```
str1: Hello World!
str2: Hello World!
str3: Copy successful
str4:GeeksForGeeks
```

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char src[] = "geeksforgeeks";

    // Here destination is not large
    // enough to store the src. so the
    // behaviour of strcpy is unspecified.
    // program may crashed, but its
    // printing geeksforgeeks
    char dest[2];

    // copying src into dest.
    strcpy(dest, src);
    printf("Copied string: %s\n", dest);

    return 0;
}
```

Output:

```
Copied string: geeksforgeeks
```

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char src[] = "geeksforgeeks";

    // The destination string size is 8
    // which is less than length of src.
    char dest[8];

    // copying 8 bytes of src into dest.
    // dest is not NULL terminated.
    strncpy(dest, src, 8);

    // using strlen function on non terminated.
    // string which can cause segfault.
    int len = strlen(dest);

    printf("Copied string: %s\n", dest);
    printf("Length of destination string: %d\n", len);

    return 0;
}
```

Output:

```
Copied string: geeksfor
Length of destination string: 8
```

8.6.1 Functions `strcpy` and `strncpy`

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

```

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

8.6.2 Functions `strcat` and `strncat`

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

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

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

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

8.7 Comparison Functions of the String-Handling Library

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

Common Programming Error 8.6

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

© 2016 Pearson Education, Ltd. All rights reserved.

8.10.2 Function `strlen`

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

© 2016 Pearson Education, Ltd. All rights reserved.

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

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

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

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

Fig. 8.35 | Using function `strlen`.

© 2016 Pearson Education, Ltd. All rights reserved.