STRINGS I

CEN116 Algorithms and Programming II

1

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.

2

Fundamentals of Strings and Characters

- 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

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

Definitions

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

  initialize a variable to the string "blue".
- The definition creates a 5-element array color containing the characters 'b', 'l', 'u', 'e' and '\0'.

5

Definitions

- 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.

6

Fundamentals of Strings and Characters

- 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, so the & is not needed with argument word.

Fundamentals of Strings and Characters

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

Fundamentals of Strings and Characters

- 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.

Character-Handling Library

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

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.

11

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.

12

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).


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

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

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

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

15

```
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.)

16

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.

Functions islower, isupper, tolower and toupper

- 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 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.)


```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.)

```
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.)

21

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

22

```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.)


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

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

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

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

```
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.)

25

Standard Input/Output Library Functions

- int getchar(void) input the next character from the standard input (keyboard) and return it as an integer.
- char *gets(char *s) Input characters from the standard input (keyboard) into the array s until a newline or end-of-file character is encountered. A terminating NULL character is appended to the array.
- int putchar(int c) Print the character stored in c.
- int puts(const char *s) Print the string s followed by a newline character.

26

References

- C How to Program, 8ed, by Paul Deitel and Harvey Deitel, Pearson, 2016.