

String

Outline

1. Introduction

- String Representation

2. String Manipulation Basics

- Declaring and initializing string variables
- Input and output of strings
- Obtaining string length
- Copying string
- Concatenating strings
- Comparing strings

1. Introduction

- A **string** is a sequence of characters.
- **String length** = total number of characters in the string.
 - e.g.,
 - Length of "**ABC DEF**" is 7
 - Length of "**\n**" is 2
 - Each of **\n** and **** represents only one character in the program; they only appear as two characters in the source code.

1.1. String Literals vs. Character Constants

- String literal
 - Enclosed by a pair of double quotes (")
 - Data type is array of char
- Character constant
 - Enclosed by a pair of single quotes (')
 - Data type is char

e.g.,

'A' – A character 'A'

"A" – A string that contains a single character 'A'

'\0' – A null character (The character with ASCII code zero)

"" – An empty string (A string that does not contain any character)

"ABC" – A string that contains three characters

1.2. String Representation in C Language

- A string is represented as an array of characters in which the first null character (i.e., the null character with the smallest index) in the array indicates the end of the string.
- e.g.,
`char s1[7] = { 'A', 'B', 'C', '\0', 'D', 'E', '\0' };`
when treated as a string, represents the string "ABC".

	A	B	C	\0	D	E	\0
Index	0	1	2	3	4	5	6

1.3. String Processing

- Processing a string is basically processing an array of characters.
- There are built-in functions to support common string operations: copying strings, comparing strings, etc.
- Pay attention!
 - A string must be terminated by a **null** character.
 - The array for storing a string must be large enough.
 - An array of size N can store a string containing up to N-1 characters. (One more space needed for the terminating null character.)

2. String Manipulation Basics

- Declaring and initializing string variables
- Input and output of strings
- Obtaining string length
- Copying string
- Concatenating string (Appending one string to another string)
- Comparing String

2.1. Declaring and initializing string variables

```
1 // All the following three variables are initialized to
2 // represent the string "Hello".
3
4 // The size of str1[] is 100; it can store a string of
5 // length up to 99 (i.e., 99 characters + 1 null character).
6 char str1[100] = "Hello";
7
8 // The size of str2[] will be set to 6 by the compiler
9 // (i.e., length of "Hello" + 1 null character).
10 char str2[] = "Hello";
11
12 // The array size is 6. With this approach, we need to
13 // manually set the terminating null character.
14 char str3[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
15
16
```


2.2. String I/O with scanf() and printf()

```
1 char str[100];
```

```
2  
3 printf("Input: ");
```

```
4 scanf("%s", str);
```

The format specifier "%s" tells scanf() to read a string made up of non-whitespace characters.

Note: There is no & before str.

```
5  
6  
7 printf("You have entered: %s\n", str);
```

The format specifier "%s" tells printf() to format and output a string.

```
8  
9  
10  
11  
12  
13 Input: Apple Orange  
14 You have entered Apple  
15  
16
```

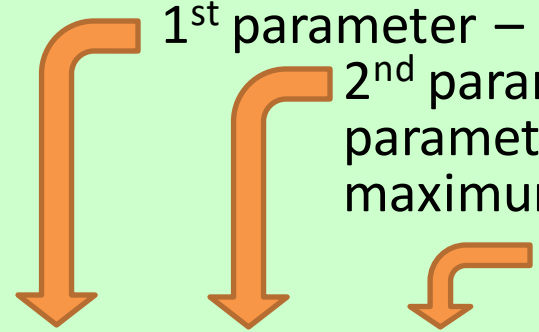
In this example, the scanf() will only read "Apple" from the input.

2.2. Reading a line of text using fgets()

```
1 char str[100];  
2  
3 printf("Input: ");  
4 fgets(str, 100, stdin);  
5  
6 printf("You have entered: %s\n", str);
```

Input: Apple Orange
You have entered Apple Orange

Here fgets() reads at most 99 characters from the input or until a newline character is encountered in the input (whichever happens first), and stores the characters (including the newline character, if any) in str[].



1st parameter – the array to store the input string
2nd parameter – the size of the array in the 1st parameter (can also be used to control the maximum number of characters to read)
3rd parameter – from where to read the input. stdin represents the console input.

fgets(str, 100, stdin);

2.3. Obtaining string length

Need to include `<string.h>` to use the built-in string related functions.

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main(void) {
5      char str1[100] = "Hello\n";
6      int i, N;
7
8      N = (int) strlen(str1);
9
10     // Print the characters in the string one by one in the
11     // reverse order
12     for (i = N-1; i >= 0; i--)
13         printf("%c", str1[i]);
14
15     return 0;
16 }
```

Think: What is the value of N in this example?

2.3. Copying strings

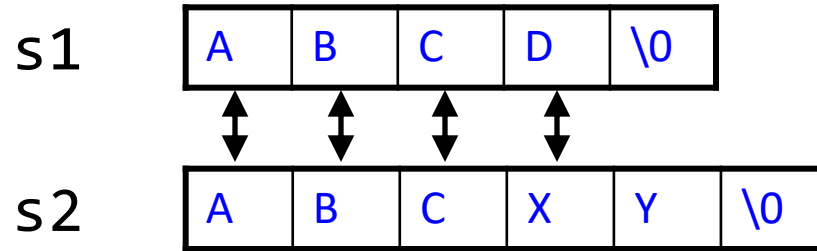
```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main(void) {
5      char str1[100], str2[100];
6
7      // Assign "Foo" to str1
8      strcpy(str1, "Foo");    // Never write this: str1 = "Foo";
9
10     // Copy the string stored in str1 to str2
11     strcpy(str2, str1);    // Never write this: str2 = str1;
12
13     return 0;
14 }
15
16
```

2.3. Concatenating string

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main(void) {
5      char str1[100] = "Hello", str2[100] = "World";
6
7      strcat(str1, " ");           // str1 becomes "Hello "
8      strcat(str1, str2);          // str1 becomes "Hello World"
9
10     return 0;
11 }
```

`strcat(str1, str2)` appends `str2` to `str1`.
`str1` must have enough space to store the concatenated string.

2.4. Strings are compared lexicographically



- Compare the 1st character of each string, then the 2nd character of each string, and so on until an order can be determined.
- ASCII value determines the order of the characters
 - ASCII value of 'X' is 88
 - ASCII value of 'D' is 68

In this example, **s1** is said to be lexicographically smaller than **s2**.

2.4. strcmp()

strcmp(s1, s2)

This function compares two strings and returns:

- A positive integer if **s1** is lexicographically larger than **s2**
- A negative integer if **s1** is lexicographically smaller than **s2**
- A zero if **s1** is equal to **s2**

2.4. String comparison

```
1 char s1[100], s2[100];
2 ... // Assume s1 and s2 are assigned some values here
3
4 // If s1 is equal to "Hello"
5 if (strcmp(s1, "Hello") == 0) {
6     ...
7 }
8
9 // If s1 is not equal to s2
10 if (strcmp(s1, s2) != 0) {
11     ...
12 }
```


2.4. String comparison (Exercise)

s1	s2	Value of strcmp(s1, s2) <i>+ve, -ve, or 0?</i>
"ABC"	"ABCDE"	
"XYZ"	"abc"	
"AB C"	"ABC"	
" "	"ABC"	
"13145"	"013145"	
"^.^"	">.<"	

2.5. Example: Lowercase to uppercase conversion

```
1  #include <string.h>
2  #include <ctype.h>
3
4  int main(void) {
5
6      char str1[100] = "Hello\n";
7      int i, N;
8
9      N = (int) strlen(str1);
10
11     for (i = 0; i < N; i++)
12         str1[i] = toupper(str1[i]);
13
14     return 0;
15 }
16
```

What is the output?

Summary

- Understand how a string is represented in the C language
- Know how to declare and initialize a string variable
- Know how to input/output strings using `scanf()`/`printf()`
- Know how to use the following string functions
 - `strlen()`, `strcpy()`, `strcat()`, `strcmp()`
- Understand how strings are being compared

Reading Assignment

- C: How to Program, 8th ed, Deitel and Deitel
- Chapter 8 C Characters and Strings
 - Sections 8.1 – 8.2: Fundamentals of Strings
 - Sections 8.4 – 8.10: String Handling Functions by Categories