hapter 2

Characters Strings

Specific Learning Outcomes

The students must have the ability to:

1. Identify and understand the char string;
2. Understand the various usage and equivalence of ASCII;
3. Formulate and apply correct solutions and coding style to laboratory programming exercises involving strings;
4. exercises involving arrays.

Learning Content

1. Representation;
2. Declaration;
3. Index of a char in a string;
4. String operations;
5. Common mistakes.

This chapter introduces the C standard library functions that help us process strings and characters. The functions enable programs to process characters, strings, lines of text and blocks of memory. The chapter discusses the techniques used to develop editors, word processors, page-layout software, computerized typesetting systems and other kinds of text processing software. The text manipulations performed by formatted input/output functions like printf and scanf can be implemented using the functions discussed in this chapter.

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. For example, 'z' represents the integer value of the letter 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 as follows:

"John Q. Doe" (a name)
"99999 Main Street" (a street address)
"Waltham, Massachusetts" (a city and state)
"(201) 555-1212" (a telephone number)

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. This is just like arrays, because strings are simply arrays of characters. A character array or a variable of type char * can be initialized with a string in a definition.

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 read-only memory.

The C standard indicates that a a string literal is immutable (i.e., 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.

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 (5) in the initializer list.

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

Common Programming Error. 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.

Error-Prevention Tip. 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.

A string can be stored in an array using scanf. For example, the following statement stores a string in character array word[20]:

```
scanf("%19s", word);
```

The string entered by the user is stored in word. Variable word is an array, which is a pointer, so the & is not needed with argument word. Recall that function 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. This prevents scanf from writing characters into memory beyond the end of the character array. (For reading input lines of arbitrary length, there's a nonstandard—yet widely supported—function readline, usually included in stdio.h.) For a character array to be printed properly as a string, the array must contain a terminating null character.

Common Programming Error. Processing a single character as a string. A string is a pointer—probably a respectably large integer. However, a character is a small integer (ASCII values range 0–255). On many systems this causes an error, because low memory addresses are reserved for special purposes such as operating-system interrupt handlers—so "access violations" occur.

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

Using Character Arrays to Store and Manipulate Strings

We have discussed only integer arrays. However, arrays are capable of holding data of any type. We now discuss storing strings in character arrays. So far, the only string-processing capability we have is outputting a string with printf. A string such as "hello" is really an array of individual characters in C.

Initializing a Character Array with a String

Character arrays have several unique features. A character array can be initialized using a string literal. For example,

```
char string1[] = "first";
```

initializes the elements of array string1 to the individual characters in the string literal "first". In this case, the size of array string1 is determined by the compiler based on the length of the string. The string "first" contains five characters plus a special string-termination character called the null character. Thus, array string1 actually contains six elements. The escape sequence representing the null character is '\0'. All strings in C end with this character. A character array representing a string should always be defined large enough to hold the number of characters in the string and the terminating null character.

Initializing a Character Array with an Intializer List of Characters

Character arrays also can be initialized with individual character constants in an initializer list, but this can be tedious. The preceding definition is equivalent to

```
char string1[] = {'f', 'i', 'r', 's', 't', '\0'};
```

Accessing the Characters in a String

Because a string is really an array of characters, we can access individual characters in a string directly using array index notation. For example, string1[0] is the character 'f' and string1[3] is the character 's'.

Inputting into a Character Array

We also can input a string directly into a character array from the keyboard using scanf and the conversion specifier %s. For example,

```
char string2[20];
```

creates a character array capable of storing a string of at most 19 characters and a terminating null character. The statement

```
scanf("%19s", string2);
```

reads a string from the keyboard into string2. The name of the array is passed to scanf without the preceding & used with nonstring variables. The & is normally used to provide scanf with a variable's location in memory so that a value can be stored there. When we discuss passing arrays to functions, we'll see that the value of an array name is the address of the start of the array; therefore, the & is not necessary. Function scanf will read characters until a space, tab, newline or end-of-file indicator is encountered. The string string2 should be no longer than 19 characters to leave room for the terminating null character. If the user types 20 or more characters, your program may crash or create a security vulnerability called buffer overflow. For this reason, we used the conversion specifier %19s so that scanf reads a maximum of 19 characters and does not write characters into memory beyond the end of the array string2. It's your responsibility to ensure that the array into which the string is read is capable of holding any string that the user types at the keyboard. Function scanf does not check how large the array is. Thus, scanf can write beyond the end of the array.

Outputting a Character Array That Represents a String

A character array representing a string can be output with printf and the %s conversion specifier. The array string2 is printed with the statement

```
printf("%s\n", string2);
```

Function printf, like scanf, does not check how large the character array is. The characters of the string are printed until a terminating null character is encountered. [Consider what would print if, for some reason, the terminating null character were missing.]

Demonstrating Character Arrays

Program 14 demonstrates initializing a character array with a string literal, reading a string into a character array, printing a character array as a string and accessing individual characters of a string. The program uses a for statement (lines 22–24) to loop through the string1 array and print the individual characters separated by spaces, using the %c conversion specifier. The condition in the for statement is true while the counter is less than the size of the array and the terminating null character has not been encountered in the string. In this program, we read only strings that do not contain whitespace characters. Notice that lines 17–18 contain two string literals separated only by whitespace. The compiler automatically combines such string literals into one—this is helpful for making long string literals more readable.

```
1 // Program_14.c
2 // Treating character arrays as strings.
3 #include <stdio.h>
4 #define SIZE 20
5
6 // function main begins program execution
7 int main(void)
8 {
9       char string1[SIZE]; // reserves 20 characters
10      char string2[] = "string literal"; // reserves 15 characters
11
12      // read string from user into array string1
13      printf("%s", "Enter a string (no longer than 19 characters): ");
14      scanf("%19s", string1); // input no more than 19 characters
15
16      // output strings
17      printf("string1 is: %s\nstring2 is: %s\n"
18      "string1 with spaces between characters is:\n",
19      string1, string2);
20
21      // output characters until null character is reached
22      for (size_t i = 0; i < SIZE && string1[i] != '\0'; ++i) {
23      printf("%c ", string1[i]);
24      }
25
26      puts("");
27 }
```

output

```
Enter a string (no longer than 19 characters): Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
```

Program 14 Treating character arrays as strings

String-Manipulation Functions of the String-Handling Library

The string-handling library (<string.h>) provides many useful functions for manipulating string data (copying strings and concatenating strings), comparing strings, searching strings for characters and other 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 Figure 2.1. Every function—except for strncpy—appends the null character to its result. [Note: Each of these functions has a more secure version described in the optional Annex K of the C11 standard.

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.

Figure 2.1

String-manipulation functions of the string-handling library

Functions strncpy and strncat specify a parameter of type size_t. 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 is more than the length of the string. 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, some implementations append null characters to the array until the total number of characters specified by the third argument are written and others stop after writing the first null character.

Common Programming Error. Not appending a terminating null character to the first argument of a strncpy when the third argument is less than or equal to the length of the string in the second argument.

Functions strcpy and strncpy

Program 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).

```
1 // Program_15.c
2 // Using functions strcpy and strncpy
3 #include <stdio.h>
4 #include <string.h>
5 #define SIZE1 25
6 #define SIZE2 15
7
8 int main(void)
9 {
10      char x[] = "Happy Birthday to You"; // initialize char array x
11      char y[SIZE1]; // create char array y
12      char z[SIZE2]; // create char array z
13
14      // copy contents of x into y
15      printf("%s%s\n%s%s\n",
16              "The string in array x is: ", x,
17              "The string in array y is: ", strcpy(y, x));
18
19      // copy first 14 characters of x into z. Does not copy null
20      // character
21      strncpy(z, x, SIZE2 - 1);
22
23      z[SIZE2 - 1] = '\0'; // terminate string in z
24      printf("The string in array z is: %s\n", z);
25 }
```

output

```
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
```

Program 15 Using functions strcpy and strncpy

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 a specified number of characters from the second string to the first string. A terminating null character is automatically appended to the result. Program 16 demonstrates function strcat and function strncat.

```
1 // Program_16.c
2 // Using functions strcat and strncat
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void)
7 {
8       char s1[20] = "Happy "; // initialize char array s1
9       char s2[] = "New Year "; // initialize char array s2
10      char s3[40] = ""; // initialize char array s3 to empty
11
12      printf("s1 = %s\ns2 = %s\n", s1, s2);
13
14      // concatenate s2 to s1
15      printf("strcat(s1, s2) = %s\n", );
16
17      // concatenate first 6 characters of s1 to s3. Place '\0'
18      // after last character
19      printf("strncat(s3, s1, 6) = %s\n", strncat(s3, s1, 6));
20
21      // concatenate s1 to s3
22      printf("strcat(s3, s1) = %s\n", );
23 }
```

output

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

Program 16 Using functions strcat and strncat

Other Functions of the String-Handling Library

The two remaining functions of the string-handling library are strerror and strlen. Figure 8.33 summarizes the strerror and strlen functions.

Function prototype	Function description
`char *strerror(int errornum);`	Maps `errornum` into a full text string in a compiler- and locale-specific manner (e.g. the message may appear in different spoken languages based on the computer's locale). A pointer to the string is returned. Error numbers are defined in `errno.h`.
`size_t strlen(const char *s);`	Determines the length of string `s`. The number of characters preceding the terminating null character is returned.

Figure 2.2

Other functions of the string-handling library

Function strerror

Function strerror takes an error number and creates an error message string. A pointer to the string is returned. Program 17 demonstrates strerror.

```
1 // Program_17.c
2 // Using function strerror
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void)
7 {
8 printf("%s\n", );
9 }
```

output

```
No such file or directory
```

Program 17 Using function strerror

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. Program 18 demonstrates function strlen.

```
1 // Program_18.c
2 // Using function strlen
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void)
7 {
8       // initialize 3 char pointers
9       const char *string1 = "abcdefghijklmnopqrstuvwxyz";
10      const char *string2 = "four";
11      const char *string3 = "Boston";
12
13      printf("%s\"%s\"%s%u\n%s\"%s\"%s%u\n%s\"%s\"%s%u\n",
14      "The length of ", string1, " is ", strlen(string1),
15      "The length of ", string2, " is ", strlen(string2),
16      "The length of ", string3, " is ", strlen(string3));
17 }
```

output

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

Program 18 Using function strlen

References

Paul Deitel, and Harvey Deitel, “C How to Program with an introduction to C++” Eight Edition

Kimberly Nelson King, “C Programming: A Modern Approach” Second Edition

Brian W. Kernighan and Dennis M. Ritchie, “C Programming Language” 2nd Edition

Jeri R. Hanly and Elliot B. Koffman, “Problem Solving and Program Design in C”, 8th Edition

https://www.thecrazyprogrammer.com

https://www.geeksforgeeks.org

https://www.thecrazyprogrammer.com

https://www.geeksforgeeks.org

Additional Resources and Links

Declaring and Initializing String Variables by Neso Academy
https://www.youtube.com/watch?v=cnfRyvo41Bs&list=PLBlnK6fEyqRhX6r2uhhlubuF5QextdCSM&index=132

Writing Strings using printf and puts Functions by Neso Academy
https://www.youtube.com/watch?v=wW7u_WrkY6Q&list=PLBlnK6fEyqRhX6r2uhhlubuF5QextdCSM&index=133

Reading Strings using scanf and gets Functions by Neso Academy
https://www.youtube.com/watch?v=2HasEQe5VR0&list=PLBlnK6fEyqRhX6r2uhhlubuF5QextdCSM&index=134

Strings (Solved Problem 1) by Neso Academy
https://www.youtube.com/watch?v=0gkOH0Ft6bk&list=PLBlnK6fEyqRhX6r2uhhlubuF5QextdCSM&index=137

C String Library and String Copy Function-strcpy() by Neso Academy
https://www.youtube.com/watch?v=DOPs6c0f4Ek&list=PLBlnK6fEyqRhX6r2uhhlubuF5QextdCSM&index=138F5QextdCSM&index=134

String Length Function-strlen() by Neso Academy
https://www.youtube.com/watch?v=DOPs6c0f4Ek&list=PLBlnK6fEyqRhX6r2uhhlubuF5QextdCSM&index=139

String Concatenate Functions-strcat() & strncat() by Neso Academy
https://www.youtube.com/watch?v=DOPs6c0f4Ek&list=PLBlnK6fEyqRhX6r2uhhlubuF5QextdCSM&index=140

String Comparison Function-strcmp() by Neso Academy
https://www.youtube.com/watch?v=DOPs6c0f4Ek&list=PLBlnK6fEyqRhX6r2uhhlubuF5QextdCSM&index=141

Array of Strings by Neso Academy
https://www.youtube.com/watch?v=DOPs6c0f4Ek&list=PLBlnK6fEyqRhX6r2uhhlubuF5QextdCSM&index=142

Strings (Solved Problem 2) by Neso Academy
https://www.youtube.com/watch?v=DOPs6c0f4Ek&list=PLBlnK6fEyqRhX6r2uhhlubuF5QextdCSM&index=143

Strings (Solved Problem 3) by Neso Academy
https://www.youtube.com/watch?v=DOPs6c0f4Ek&list=PLBlnK6fEyqRhX6r2uhhlubuF5QextdCSM&index=144

Strings (Solved Problem 4) by Neso Academy
https://www.youtube.com/watch?v=DOPs6c0f4Ek&list=PLBlnK6fEyqRhX6r2uhhlubuF5QextdCSM&index=145

Strings (Solved Problem 5) by Neso Academy
https://www.youtube.com/watch?v=DOPs6c0f4Ek&list=PLBlnK6fEyqRhX6r2uhhlubuF5QextdCSM&index=146

Strings (Solved Problem 6) by Neso Academy
https://www.youtube.com/watch?v=DOPs6c0f4Ek&list=PLBlnK6fEyqRhX6r2uhhlubuF5QextdCSM&index=147

Chapter Assessment

Answer the following.

I. Show two different ways to initialize character array vowel with the string of vowels "AEIOU".

II. What, if anything, prints when each of the following C statements is performed? If the statement contains an error, describe the error and indicate how to correct it. Assume the following variable definitions:

```
char s1[50] = "jack", s2[50] = "jill", s3[50];
```

a. printf("%c%s", toupper(s1[0]), &s1[1]);
b. printf("%s", strcpy(s3, s2));
c. printf("%s", strcat(strcat(strcpy(s3, s1), " and "), s2));
d. printf("%u", strlen(s1) + strlen(s2));
e. printf("%u", strlen(s3)); // using s3 after part (c) executes

III. Find the error in each of the following program segments and explain how to correct it:

a.
```
char s[10];
strncpy(s, "hello", 5);
printf("%s\n", s);
```
b.
```
printf("%s", 'a');
```
c.
```
char s[12];
strcpy(s, "Welcome Home");
```
d.
```
if (strcmp(string1, string2)) {
puts("The strings are equal");
}
```

Notes

Notes

Notes