

PROGRAMMING METHODOLOGY

Lab 5: Character and String

1 Introduction

In this lab tutorial, we'll introduce the C standard library functions that manipulating string and character. These functions enable our program to process characters, strings, lines of text and block of memory.

2 Fundamental of String and Character

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. For example, to represent a name, we declare as "John Smith", or a street address is "19 Nguyen Huu Tho Street".

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** or a **variable of type `char*`** can be initialized with a string in a definition. Each declaration initializes a variable to the string "blue". The first declaration creates an array with 5 elements, containing the characters: 'b', 'l', 'u', 'e' and '\0'. The second declaration creates a pointer variable `clrPtr` that points to the string "blue" somewhere in memory.

```
char color[] = "blue";  
char *clrPtr = "blue";
```

Printing a string that does not contain a terminating null character, `'\0'`, is an error. 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.

3 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. The following program will present an approach to read and assign an input string to array variable. We use functions **fgets**, **scanf** and **putchar** to read a line of text from the standard input (keyboard) and then, return the number of characters (string's length). Function **fgets** reads characters from the standard input into its first argument – an array of chars – 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. 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.

```
1  #include <stdio.h>
2  #define SIZE 100
3
4  int getLength(const char*);
5
6  int main()
7  {
8      char str[SIZE]; // create char array
9
10     printf("Enter your name: ");
11     fgets(str, SIZE, stdin);
12
13     printf("String: %s\n", str);
14     printf("String's length: %d", getLength(str));
15
16     return 0;
17 }
18
19 int getLength(const char* str)
20 {
21     if(str[0] == '\0') return 0;
22
23     int length = 0;
24     while(str[length] != '\0')
25     {
26         length = length + 1;
27     }
28
29     return length - 1; // exclude new line character
30 }
```

4 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). Characters are often manipulated as integers, because a character in C is a one-byte integer. Table below describes some useful functions to manipulate a character.

Prototype	Description
<code>int isblank(int c);</code>	Returns a true value if <i>c</i> is a <i>blank character</i> that separates words in a line of text and 0 (false) otherwise.
<code>int isdigit(int c);</code>	Returns a true value if <i>c</i> is a <i>digit</i> and 0 (false) otherwise.
<code>int isalpha(int c);</code>	Returns a true value if <i>c</i> is a <i>letter</i> and 0 otherwise.
<code>int isalnum(int c);</code>	Returns a true value if <i>c</i> is a <i>digit</i> or a <i>letter</i> and 0 otherwise.
<code>int islower(int c);</code>	Returns a true value if <i>c</i> is a <i>lowercase letter</i> and 0 otherwise.
<code>int isupper(int c);</code>	Returns a true value if <i>c</i> is an <i>uppercase letter</i> and 0 otherwise.
<code>int tolower(int c);</code>	If <i>c</i> is an <i>uppercase letter</i> , <code>tolower</code> returns <i>c</i> as a <i>lowercase letter</i> . Otherwise, <code>tolower</code> returns the argument unchanged.
<code>int toupper(int c);</code>	If <i>c</i> is a <i>lowercase letter</i> , <code>toupper</code> returns <i>c</i> as an <i>uppercase letter</i> . Otherwise, <code>toupper</code> returns the argument unchanged.

The program below illustrates these functions.

```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  int main()
5  {
6      printf("Is '8' digit? - %d\n", isdigit('8'));
7      printf("Is 'A' digit? - %d\n", isdigit('A'));
8      printf("Is 'a' in lower-case? - %d\n", islower('a'));
9
10     printf("To lower-case of 'A' - %c\n", tolower('A'));
11     printf("To upper-case of 'a' - %c\n", toupper('A'));
12
13     return 0;
14 }
```

5 String-Handling Library

The string-handling library (`<string.h>`) provides many useful functions for manipulating string data: copying strings, concatenating strings, and comparing strings. Table below describes some popular function in `<string.h>` library.

<i>Prototype</i>	<i>Description</i>
<code>char *strcpy (char *s1, const char *s2)</code>	Copies string s_2 into array s_1 . The value of s_1 is returned.
<code>char *strncpy (char *s1, const char *s2, size_t n)</code>	Copies at most n characters of string s_2 into array s_1 . The value of s_1 is returned.
<code>char *strcat (char *s1, const char *s2)</code>	Appends string s_2 to array s_1 . The first character of s_2 overwrites the terminating null character of s_1 . The value of s_1 is returned.
<code>char *strncat (char *s1, const char *s2, size_t n)</code>	Appends at most n characters of string s_2 to array s_1 . The first character of s_2 overwrites the terminating null character of s_1 . The value of s_1 is returned.
<code>int strcmp (const char *s1, const char *s2)</code>	Compares the string s_1 with the string s_2 . The function returns 0, less than 0 or greater than 0 if s_1 is equal to, less than or greater than s_2 , respectively.
<code>int strncmp (const char *s1, const char *s2, size_t n)</code>	Compares up to n characters of the string s_1 with the string s_2 . The function returns 0, less than 0 or greater than 0 if s_1 is equal to, less than or greater than s_2 , respectively.

The program below illustrates these functions.

```

1  #include <stdio.h>
2  #include <string.h>
3  #define SIZE 100
4
5  int main()
6  {
7      char str[] = "Happy New Year";
8      char y[] = " 2017";
9      char x[SIZE];
10
11     strcpy(x, str);
12     printf("x is copied: %s\n", x);
13
14     printf("Compare x and str: %d\n", strcmp(str, x));
15
16     printf("Concatenate y to x: %s\n", strcat(x, y));
17
18     return 0;
19 }
```

6 Exercises

1. Input a string and return the string's length.
2. Input a string and print it in reverse order.

3. Input a string represent a full name, split and print the first name and the last name.
4. Input a string and normalize it (trim the space before, inside, and after; to lower each token except the first). For example, with the input " PrOgRaMmInG MeThOd ", the output will be "Programming Method".
5. Input two string s_1 and s_2 , concatenating s_2 to s_1 . (Note: don't use *string.h* library)
6. Input a string and check whether a character appears or not. If yes, return the first position.
7. Input a string and check whether a character appears or not. If yes, return the all appearing positions.
8. Input a string and check whether a word appears or not. If yes, return the first position.
9. Input two string s_1 and s_2 , return the first position where s_2 appears in s_1 .
10. Input two string s_1 , s_2 , and position. Then insert s_2 to s_1 from this position.
11. Input a string and two integer numbers, named n and *position*. Then delete n characters from *position* of input string.

7 Reference

- [1] Brian W. Kernighan & Dennis Ritchie (1988). *C Programming Language, 2nd Edition*. Prentice Hall.
- [2] Paul Deitel & Harvey Deitel (2008). *C: How to Program, 7th Edition*. Prentice Hall.
- [3] *C Programming Tutorial* (2014). Tutorials Point.
- [4] *C Programming* (2013). Wikibooks.