Chapter 2

Strings

2.1 Motivation

The document that you are reading right now is a collection of letters and other characters (such as period, left parentheses, question mark, and brackets). Letters when combined properly form words; words form phrases and sentences; sentences form paragraphs.

Text editors and wordprocessors are software that we use for editing, storing, and manipulating characters. Web browsers, email utilities, messengers, chat programs are just some of the other software tools that we use to create, store, retrieve and view information encoded as characters.

Aside from numeric data, a programmer will have to learn how to represent, store and manipulate a collection of characters commonly referred to as *string*.

2.2 What is a string?

Oh yeah, we can have zero character strings :O.

A *string* is a combination of zero or more characters from a given character set.[1] For example, `"Hello world!"` is a string constant.

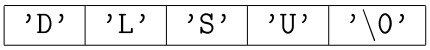
A string constant is denoted in the C programming language by writing a sequence of characters enclosed within a pair of double quotes. The following are examples of string constants:

[1] We will use the ASCII characters.

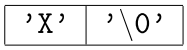
```
"ABC"
"DLSU"
"COMPRO2"
"Hello world!\n"
"http://www.abc_def_123.org"
"X"
" "
""
```

2.3 String Representation

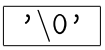
A string is internally represented as a sequence of characters and stored in contiguous bytes of memory space. For example, the string `"DLSU"` is stored internally as follows



The string `"X"` is stored internally as



and the empty string `""` is stored as



The last character in a string is always a *null byte* denoted by `'\0'` (backslash character followed by zero). The null byte has an ASCII code of 0. It is a hidden character so it will not be visible on the display screen or printout. Notice that the `'\0'` is not explicitly written as part of the string constant. It will be automatically appended in behalf of the programmer or user.

The *length* of a string is the number of characters in the string *excluding* the null byte. For example, the length of the string `"COMPRO2"` is 7. Notice that the string `" "` contains a space character and its length is 1. It is different from the string `""` which is used to denote an *empty* string. An empty string has a length of 0. It should also be noted that the string constant `"X"` is different from the character constant `'X'`.

2.4 Memory Allocation for Strings

While there is an `int` keyword corresponding to an integer data type, there is no `str` keyword because the C programming language does not explicitly provide a string data type. Instead, it uses contiguous bytes for storing the individual elements of a string.

The storage space can be allocated by declaring a one–dimensional array of characters (static memory allocation) or via `malloc()` (dynamic memory allocation). We will limit the discussion in this chapter to character arrays for simplicity reason.

Let us assume for example that we would like to allocate memory space for storing at most 3 characters *excluding* the null byte. We can achieve this by declaring a character array as shown in the following code:

```
char str[4];  /* static mem. allocation */
```

Notice that the total size is 4 bytes (not 3) in order to store the null byte. In general, if the length of the string is n then we should allocate memory space for at least $n + 1$ number of bytes.

2.5 String Initialization

In Chapter 2, we learned how to define an array of characters. For example, we can define the value of the character array `str` as `"ABC"` by

```
char str[4] = {'A', 'B', 'C', '\0'}; /* variable definition */
```

An equivalent but more compact initialization can be achieved as shown in the following definition:

```
char str[4] = "ABC";  /* variable definition */
```

This form of initialization is the preferred method and the one that we will use from hereon. Its internal representation is as follows.

'A'	'B'	'C'	'\0'

If the array size is more than the length of the string constant, the unused bytes will be automatically initialized to ASCII 0. For example, the definition:

```
char str[10] = "ABC";  /* variable definition */
```

is represented internally as:

'A'	'B'	'C'	0	0	0	0	0	0	0

It should be noted that the use of the assignment operator `=` with a string constant is valid only if done as a variable definition.[2] The following assignment operation is not allowed and will cause a syntax error.

```
char str[4];  /* note: this is a variable declaration only */

str = "ABC";  /* syntax error! */
```

► Self–Directed Learning Activity ◄

Encode and compile the following program.

```
int main()
{
    char str[4] = "ABC";
    char name[5];

    return 0;
}
```

1. Decrease the size of array `str` from 4 to 2. Compile the program again. Is there any compilation error? If yes, what is the nature of the error? What can you conclude based on this experiment?

2. Change the size of array `str` back to 4. Insert the assigment statement `name = "JUAN";` just before the `return` statement. Compile the program. Is there any compilation error? If yes, what is the nature of the error?

[2]Recall that variable declaration does not involve an initialization unlike a variable definition.

2.6 String I/O with scanf() and printf()

The pre–defined functions `printf()` and `scanf()` have already been introduced in COMPRO1. They are also used, respectively, for output and input of strings with `"%s"` as formatting symbol. Oh, really now, huh...
So strings have their corresponding format specifier "%s".

Listing 2.1 shows an example of string I/O with `printf()` and `scanf()`. Notice in particular that it is incorrect to put an address–of operator (ampersand) before `str` in the `scanf()`. Recall that the name of the array is synonymous to the address of the first element.
That makes complete sense!
Array names are addresses to the first element they contain, which removes the need for an ampersand.

Listing 2.1: `printf()` and `scanf()` with strings

```
1  #include <stdio.h>
2  int main()
3  {
4      char prompt[30] = "Input a word (max. of 10 characters): ";
5      char str[11];
6
7      printf("%s", prompt);
8      scanf("%s", str);
9      printf("You entered %s\n", str);
10
11     return 0;
12 }
```

Really now?
I thought we had to dereference it first because it was a pointer.

▶ Self–Directed Learning Activity ◀

1. Encode and run the program in Listing 2.1. Try the following as inputs (do not include a pair of double quotes):

   - `X`
   - `test`
   - `abc@def`
   - `COMPRO2`
   - `VeryLongWord`
   - `brown fox`

   Note that there is no space in the fifth item, but there is a space in the last item. What are the corresponding results?

Try other inputs. Try it with shorter than 10 characters, with exactly 10 characters, with more than 10 characters and with inputs that include spaces.

(a) What happens when the number of characters is more than the size of array `str`?

(b) What happens when the user inputs something that has a space in between?

2. A run–time error occurs if the user inputs characters more than the size of the character array. To limit the number of characters accepted by the `scanf()` function, specify the limit as a positive integer between `%` and `s`. For example, to limit the maximum number of input characters to 10, the format should be `"%10s"`. Modify and test the program above to verify that this is true.

3. Modify the program above by putting an ampersand immediately before variable `str` in `scanf()`. Compile and run the program. What is the result? What is the cause of the error?

4. Hangman is a game that allows the player to guess the letters in a word, i.e. a string. Try playing a Flash Hangman game at `http://www.manythings.org/hmf/`. How would you implement your own Hangman game in C? (do not think of the graphical user interface at this point in time).

2.7 String and `char *`

The individual elements of the array defined, for example, as

```
char str[4] = "ABC";
```

can be accessed using array indexing notation. The following equality relationship hold.

```
str[0] == 'A'
str[1] == 'B'
str[2] == 'C'
str[3] ==  '\0'
```

The address of each byte storing a character value has a data type of `char *`. Moreover, in the C programming languge, the name of the array is synonymous with the address of the first array element. Thus, `str[i] == *(str + i)`. This means that the following equality relationship also hold.

```
*(str + 0) == 'A'
*(str + 1) == 'B'
*(str + 2) == 'C'
*(str + 3) ==  '\0'
```

2.8 String Manipulation Functions

There are several pre–defined functions for manipulating strings. Their function prototypes can be found in the header file `string.h`. In this document, we will learn four of the most commonly used string manipulation functions.

The function prototype that we will encounter in the following subsections pass the base address (i.e., address of the first byte in a character array) as parameter. In the actual function call, we simply need to supply the name of the array.

2.8.1 Determining the Length of a String

The `strlen()` function is used to determine the length of a string.[3] Its function prototype is as follows:

```
size_t strlen(const char *str)
```

Listing 2.2 shows an example on how to use the `strlen()` function.

Listing 2.2: Example program illustrating `strlen()`

```
1 #include <stdio.h>
2 #include <string.h>  /* don't forget to include this file */
3
4 int main()
5 {
```

[3]Remember that the length of the string does not include the null byte.

```
6       char str[4] = "ABC";
7       char name[21];
8
9       printf("The length of str is %d\n\n", strlen(str));
10      printf("Input your name (max. of 20 chars.): ");
11      scanf("%20s", name);
12      printf("The length of your name is %d characters.\n",
13              strlen(name));
14
15      return 0;
16  }
```

Encode and run the program to see how it works. Experiment by trying strings of different lengths.

Listing 2.3 shows an example program that asks the user to input a string. Thereafter, it prints the input string one character per line.

Listing 2.3: Another example illustrating `strlen()`

```
1   #include <stdio.h>
2   #include <string.h>  /* don't forget to include this file */
3
4   int main()
5   {
6       char data[21];
7       int i;
8       int len;
9
10      printf("Input a string (max. of 20 chars.): ");
11      scanf("%20s", data);
12
13      len = strlen(data);
14      for (i = 0; i < len; i++)
15          printf("data[%d] = %c\n", i, data[i]);
16
17      return 0;
18  }
```

2.8.2 Copying a String

The `strcpy()` function is used to copy the value of a string (including the null byte) from a source to a destination variable. Its function prototype is shown below.

```
char *strcpy(char *str1, const char *str2)
```

Here, `str2` is the pointer to the source string and `str1` is the pointer to the destination. The function returns the pointer to the source string `str1`. In actual programming practice, the return value is ignored.

We already know that the assignment statement `str = "ABC";` is incorrect. The correct initialization is done via `strcpy(str, "ABC");` as shown in Listing 2.4. Note that we can simply ignore the function's return value.

Listing 2.4: Example program illustrating `strcpy()`

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main()
5  {
6      char source[8] = "COMPRO2";
7      char str[4];
8      char name[21];
9      char destination[8];
10
11      /* copy a string constant */
12     strcpy(str, "ABC");   /* str = "ABC"; is incorrect! */
13     printf("The value of str = %s.\n", str);
14
15     /* copy a string constant with spaces in between */
16     strcpy(name, "Juan dela Cruz");
17     printf("The value of name = %s.\n", name);
18
19     /* copy the value of a string variable */
20     strcpy(destination, source);
21     printf("The value of destination = %s.\n", destination);
22
23     return 0;
24 }
```

2.8.3 Concatenating Strings

The `strcat()` function is used to concatenate two strings. Its function prototype is given below.

```
char *strcat(char *str1, const char *str2)
```

Here `str1` and `str2` are pointers to the first and second strings respectively. Characters from the second string are copied onto the first string starting at the memory space associated with the first string's null byte. We should ensure that the size of `str1` is big enough to accomodate its original characters and the characters from `str2`; otherwise, the result is undefined. The second string remains the same after calling `strcat()`.

Listing 2.5 shows an example program how to use `strcat()`.

Listing 2.5: Example program illustrating `strcat()`

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main()
5  {
6      char str1[4] = "ABC";
7      char str2[4] = "DEF";
8      char string[20];
9
10     /* initialize as empty string */
11     strcpy(string, "");
12
13     /* concatenate empty string "" and "Hello" */
14     strcat(string, "Hello");
15     printf("string = %s\n", string);
16
17     strcpy(string, "");
18     strcat(string, str1);
19     strcat(string, " "); /* append space */
20     strcat(string, str2);
21     printf("string = %s\n", string);
22
23     return 0;
24 }
```

2.8.4 Comparing Strings

The `strcmp()` function is used to compare two strings lexicographically. It was made available because we cannot use the relational operators such as `==`, `>` and `<` to compare strings. Its function prototype is given below.

```
int strcmp(const char *str1, const char *str2)
```

There are three possible return values, namely:

- 0 is returned when the strings are equal

  For example, `strcmp("ABC", "ABC")` returns 0.

- a negative value is returned if the first string is less than the second string

  For example, `strcmp("ABC", "XYZ")` returns a negative value.

- a positive value is returned if the first string is greater than the second string

  For example, `strcmp("XYZ", "ABC")` returns a positive value.

Note that the strings need not be of the same length. For example, the function call `strcmp("Hey", "Jude")` returns a negative value.

A example program illustrating how to use `strcmp()` is shown in Listing 2.6.

Listing 2.6: Example program illustrating `strcmp()`

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main()
5  {
6      char str1[4] = "ABC";
7      char str2[4] = "XYZ";
8
9      /* "ABC" is equal to "ABC", result is 0 */
10     printf("%d\n", strcmp(str1, "ABC"));
11
12     /* "ABC" is less than "XYZ", result is a negative value */
13     printf("%d\n", strcmp(str1, str2));
14
15     /* "XYZ" is greater than "ABC", result is a positive value */
```

```
16      printf("%d\n", strcmp(str2, str1));
17
18      /* "ABC" is less than "X", result is a negative value */
19      printf("%d\n", strcmp("ABC", "X"));
20
21      /* "aBC" is greater than "ABC", result is a positive value */
22      printf("%d\n", strcmp("aBC", "ABC"));
23
24      return 0;
25  }
```

2.9 Strings and typedef

The C programming language has a keyword `typedef` for creating a synonym or alias of a specified data type name. The syntax for declaring an alias is

```
typedef <type name> <alias>;
```

Once declared, the alias can be used in place of the original type name.

The following example declares an alias for an `int` type called `Boolean`. Thereafter, the `Boolean` alias is used to declare `flag` as a variable that will be limited by the programmer to a value of either `FALSE` or `TRUE`.

```
#define FALSE (0)
#define TRUE (!(FALSE))

typedef int Boolean;

int main()
{
    Boolean flag = FALSE;

    /* -- other statements follow --*/
}
```

The `typedef` is not limited to the basic data types. It can be used together with pointer data type, with array and as well as `struct` data type.[4]

[4] `struct` data type will be discussed in Chapter 4.

We have already mentioned the fact that the C programming language does not have a data type for string. An artificial way to support a string data type is to use `typedef` with a character array as shown in the following example code.

```
typedef char String[51];

void test(String str)
{
     /*-- some statements here --*/
}

int main()
{
     String sentence = "Hello world!";
     String greetings;

     test(sentence);

      /* -- other statements follow --*/

     return 0;
}
```

The program defines `String` as a global alias for a character array of size 51. Thereafter, `sentence` and `greetings` were defined and declared as variables of "type" `String` respectively in the `main()` function. The alias can also be used as parameter type as shown in the `test()` function definition.

In actual programming practice, it is better if we specify the maximum size for the string as part of the alias name. In the example program shown in Listing 2.7, the alias `String20` was defined and later used as the data type for variables `lastname` and `firstname`. The number 20 indicated as part of the alias name will help the programmer remember that at most 20 characters (excluding the null byte) can be stored in the associated variable. Notice that the actual size of the character array in the `typedef` is 20 + 1. The last byte ensures that there is a space for the null byte in case all the first 20 bytes are used for non–zero characters.

Listing 2.7: Example program illustrating `typedef`

```
1  #include <stdio.h>
2  #include <string.h>
3
4  typedef char String20[21];
5  typedef char String50[51];
6
7  void Input(String50 prompt, String20 str)
8  {
9       printf("%s", prompt);
10      scanf("%20s", str);
11 }
12
13 void Greet(String20 lastname, String20 firstname)
14 {
15      String50 greet;
16
17      strcpy(greet, "Hello ");  /* note space after '0' */;
18      strcat(greet, firstname);
19      strcat(greet, " ");
20      strcat(greet, lastname);
21      strcat(greet, "! How are you today?");
22
23      printf("%s\n", greet);
24 }
25
26 int main()
27 {
28     String20 lastname;
29     String20 firstname;
30
31     Input("Enter your last name: ", lastname);
32     Input("Enter your first name: ", firstname);
33     Greet(lastname, firstname);
34
35     return 0;
36 }
```

2.10 Array of Strings

The need to represent, store and manipulate a collection of strings arise in several programming problems. For example, how can we store the 32 keywords in the ANSI C programming language? One way to do this is to declare 32 string variables each with a distinct name, and initialize them with the name of the C keyword. For example:

```
typedef char String10[11];

String10 keyword0;
String10 keyword1;
      :
      :  /* more variable declarations */
      :
String10 keyword31;
```

This is a brute force approach and should be avoided in actual practice.

The recommended practice is to declare an array of strings. Listing 2.8 shows an example of how this is done. We first declare an alias for a string, and thereafter, we declare (or define) an array of string.

Listing 2.8: Array of strings example program 1

```
1  #include <stdio.h>
2
3  typedef char String10[11];
4
5  int main()
6  {
7      /* keywords is declared/defined as an array of String10.
8       We show the initialization for the 1st 5 keywords only. */
9      String10 keywords[32] = {"auto", "break", "case",
10                              "char", "const"};
11     int i;
12
13     printf("The C keywords are:\n\n");
14     for (i = 0; i < 5; i++)
15         printf("%s\n", keywords[i]);
16
17     return 0;
18 }
```

In this example, variable `keywords[]` is an array of size 32. Each element of the array, i.e., `keywords[i]` is of type `String10`.

Listing 2.9 shows another example. The alias `String10` is first declared. Thereafter, it is used to declare variable `friends[]` as an array of size 5 with each element of type `String10` in `main()`. The program also demonstrates how to pass the name of the array of strings as function parameter as shown in `InputNicknames()` and `PrintNicknames()` functions.

Listing 2.9: Array of strings example program 2

```
1  #include <stdio.h>
2
3  typedef char String10[11];
4
5  void InputNicknames(String10 friends[], int n)
6  {
7       int i;
8
9       for (i = 0; i < n; i++) {
10           printf("Input the nickname of your friend: ");
11           scanf("%s", friends[i]);
12      }
13 }
14
15 void PrintNicknames(String10 friends[], int n)
16 {
17     int i;
18
19     printf("\n");
20     printf("The nicknames of your friends are:\n");
21     for (i = 0; i < n; i++) {
22          printf("%s\n", friends[i]);
23     }
24 }
25
26 int main()
27 {
28     String10 friends[5];
29
30     InputNicknames(friends, 5);
31     PrintNicknames(friends, 5);
32
33     return 0;
34 }
```

2.11 Chapter Summary

The key ideas presented in this chapter are summarized below:

- A string is a combination of zero or more characters from a given character set.
- A string constant is specified by enclosing a group of characters enclosed within a pair of double quotes.
- A null byte indicates the end of a string. It is used by string manipulation functions such as `strlen()` and `strcat()` to determine the last character within the string.
- The data type associated with the address of each string element is `char *`.
- Memory space for a string can be allocated as a one dimensional array of characters.
- `printf()` can be used to output a string with `"%s"` as format.
- `scanf()` can be used to input the value of a string with `"%s"` as format. The maximum number of characters in the input can be indicated by putting the limiting value between `"%"` and `"s"`.
- Pre–defined functions for string manipulations exist with their function prototypes declared in the file `stdio.h`.
- `typedef` can be used to declare an alias for a character array.
- An array of strings can be represented, stored and manipulated by first declaring an alias for the character array, and then use the said alias as a data type for declaring an array of strings.

Problems for Chapter 2

Problem 2.1. Assume the following declarations and definitions:

```
typedef char String7[8];
typedef char String30[31];

String7 subject1 = "COMPRO2";
String7 subject2 = "FORMDEV";
String30 sentence;
```

What is output corresponding to the following? Items are independent from each other.

```
a.  printf("%d\n", strlen("Hello World!"));
b.  printf("%d\n", strlen("X"));
c.  printf("%d\n", strlen("X\n"));
d.  printf("%d\n", strlen(" "));
e.  printf("%d\n", strlen(""));
f.  printf("%d\n", strlen(strcpy(sentence, "Nanja kore???")));
g.  printf("%d\n", strlen(strcat(strcpy(sentence, "A*B*"), "C*D*F")));
h.  printf("%d\n", strcmp(subject1, subject2));
i.  printf("%d\ns, strcmp(subject2, subject1));
j.  printf("%d\n", strcmp("formdev", subject1));
k.  printf("%d\n", strcmp(subject2, "compro2"));
l.  printf("%d\n", strcmp(subject2, "formdev"));
```

Problem 2.2. Implement `void PrintReversed(char *str);` which will output the string in reversed order. For example, the function call `PrintReversed("ABC");` will output `"CBA"`.

Problem 2.3. Implement `int IsPalindrome(char *str);` which will return a 1 if the string parameter is a palindrome otherwise it returns a 0. A palindrome is a word or phrase that when read backwards (or in reversed order) produces the same word or phrase. For example, the word `"ROTOR"` is a palindome. The phrase `"STAR RATS"` is a palindrome. The word `"GOOD CAT"` is not a palindrome.

Problem 2.4. Implement `char *Capitalize(char *str);` which will capitalize all lower case letters in the string. The functions returns the pointer to the first byte of the modified string. For example, let `str1` and `str2` be strings. Also, let `str1` contain `"Hello World!"`. The function call `strcpy(str2, Capitalize(str1));` will copy `"HELLO WORLD!"` as the value of variable `str2`.

Problem 2.5. Implement a function `int GetPassword(char *password)` that will do the following in sequence:

1. Ask the user to input a string which will be stored into a local variable named `str` (maximum of 20 characters).
2. Check if `str` is equivalent to `password`. If it is, the function returns a 1. (Note: user entered the correct password.).
3. If they are not equivalent, i.e., the password is incorrect, repeat from step 1. The user is given three chances to enter the correct password. If after three tries, the correct password was not supplied, the function will return a value of 0.

Problem 2.6. The program in Listing 2.8 initializes the array of strings named as `keywords[]` with the first 5 C keywords only. Modify the program by supplying the remaining 27 keywords; see `http://www.cprogrammingreference.com/Tutorials/Basic_Tutorials/ckeywords_home.html` for the complete list. Thereafter, modify the `for` loop such that all the 32 keywords will be displayed on the screen.

Problem 2.7. Refer to Listing 2.9. The task in this problem is to implement the function `void SortNicknames(String10 friends[], int n)` which will sort the `friends[]` array alphabetically (in increasing order). Use the straight selection sorting algorithm discussed in the previous chapter. Test the function by calling it inside `main()` immediately before the call to `PrintNicknames()` function.

Problem 2.8. Create and implement your own algorithms for the following:

- String length, function prototype is: `int mystrlen(char *str);`
- String copy, function prototype is: `char *mystrcpy(char *str1, char *str2);`
- String concatenation, function prototype is: `char *mystrcat(char *str1, char *str2);`
- String comparison, function prototype is: `int mystrcmp(char *str1, char *str2);`