Unit 16: Strings

Learning Objectives

After this unit, students should:

- understand that a string is an array of chars
- know that a string always ends with a `char` with value 0 (or `'\0'`)
- be aware of the different types of special chars
- understand what the empty string represents
- be able to use the CS1010 library functions to read strings and arrays of strings

Just a `char` Array

We have seen strings as a sequence of characters stored in double-quotes, e.g., "hello!". In C, a string is nothing more than just an array of `char` values (Recall from Unit 5 that `sizeof char` is 1).

The only thing special about a string is that it *always* end with a `0` value (note: not character '0' which has a value of 48, but the value 0). Since the character with value 0 is called the null character, written as `'\0'`, we refer that strings in C as null-terminated strings.

In C, we distinguish between a string and a `char` by the quotes used. String uses double quotes `"`, while a `char` uses a single quote `'`. So we can do the following and they are equivalent:

```
1  char hello1[7] = {'h', 'e', 'l', 'l', 'o', '!', 0};
2  char hello2[7] = "hello!";
```

Of course, nobody initializes a string in the first way above.

We can also skip the size of the array, as mentioned above, or, more commonly, use a `char *` type for a variable storing a constant string.

```
1  char hello3[] = "hello!";
2  char *hello4 = "hello!";
```

Special Characters

The null character is written as `'\0'`. The use of the backslash `\` creates an escape sequence that can be used to denote characters that would otherwise not visible on the screen. For instance, besides `'\0'`, we will likely encounter `'\n'` (the newline character) `'\t'` (the tab character) and `'\a'` (the beep character) regularly.

Furthermore, since we already use `\` to indicate the escape sequence, `'` to indicate a character, and `"` to indicate a string, to use these characters, we need to "escape" them -- we use `'\\'` for the backslash character, `'\''`, the single quote character, and the `'\"'` double-quote character.

String Literals

A *string literal* refers to a string written between two `"` characters, such as `"Hello world!"`. Such a string is still internally stored as an array of `char` values. The location these arrays are stored depends on the platform, but usually, they are stored in a read-only region of the memory called the `text` region. These strings are not meant to be modified. Hence, trying something like this:

```
1  char *str1 = "Hello!";
2  str1[5] = '.';
```

is not allowed and would crash your program.

The following, however, is OK:

```
1  char str2[7] = "Hello!";
2  str2[5] = '.';
```

The difference between the two is that `str1` points to a read-only region in the memory, while `str2` contains a copy of the string on the stack.

Empty String

We commonly use the empty string `""` to indicate a special condition or to initialize a string variable, where appropriate. The empty string is nothing but an array where the 0-th element is '\0'.

Common Bugs

A very common bug for beginners to C programming is to forget about the terminiating `'\0'` char in a string. For instance, suppose you want to copy one string to another:

```
    char src[6] = "hello!"
    char dst[6];
    for (long i = 0; i < 6; i += 1) {
        dst[i] = src[i];
    }
```

The string variable `dst` is not terminated and C would treat whatever follows the memory location as part of the string!

The other common bug is to create an array that is too small for the string. Suppose you use the function `strcpy` provided by C to copy the string instead.

```
    char src[6] = "hello!"
    char dst[6];
    strcpy(dst, src);
```

`strcpy` would properly copy the characters `'h'`, `'e'`, .. `'!'`, AND the terminating `'\0'`. That's a total of 7 characters. The code, however, only allocated the space of 6 characters for `dst`. So overflow occurs and your program might crash.

So, remember to always account for the "invincible friend" at the end of every string, and leave a space for it.

CS1010 I/O Library

The CS1010 I/O library provides two functions, one to read a word (separated by white-space characters) and the other to read a line (separated by a newline character). They are `cs1010_read_word()` and `cs1010_read_line()` respectively. We can also read multiple words and multiple lines with `cs1010_read_word_array()` and `cs1010_read_line_array()`. The results are stored in an array of strings.

Problem Set 16

Problem 16.1

Write the following functions (without calling the standard C functions declared in `<string.h>` such as `strlen`, `strcmp`, `strstr`):

(a) `long string_length(char *str)` return the length (i.e., the number of characters) of the string `str`.

(b) `bool string_equal(char *str1, char *str2)` return `true` if the two strings `str1` and `str2` contains exactly the same content, `false` otherwise. (Note: `str1 == str2` does not compare if two strings have the same content. (Why?))

(c) `char *string_in_string(char *needle, char *haystack)` returns a pointer to the first character of the first occurrence of `needle` in `haystack` if found. If `needle` does not occur anywhere in `haystack`, return NULL. If `needle` is an empty string, `haystack` is returned.