C strings (also called *null-terminated strings*)

fonte del testo originale: https://www.prismnet.com/~mcmahon/Notes/strings.html

Declaration and initialization

A **C string** is an array of `char` containing zero or more characters, followed by a **null character** `'\0'`. Note that, just as the newline character `'\n'`, in C the null character `'\0'` is a *single* character, even if it is represented graphically by the two consecutive symbols \ and 0. An array of `char` is **NOT** by itself a C string:

```
char string1[20];
/* this array can hold a C string, but is not yet a valid C string
   because it does not contain the terminating null character */
```

A consequence of the use of the terminating character is that the array must be at least **one more character** long than the number of characters it has to contain.

To insert text into a C string, you have to do it one array element at a time (just as you do with any other array in C): for instance, using a cycle. There is also a function, `strcpy` (string copy), which allows copying all characters of a text into a C string with a single instruction:

```
strcpy(string1, "some text");
/* this inserts "some text" followed by a '\0', into the array string1 */
```

To be able to use `strcpy` in your program, you must use directive

```
#include <string.h>
```

because `strcpy` is defined in library `string.h`.

In C, text between double quotes is a **string constant**. You can use a string constant everywhere a C string is required: for instance, if you want to print the string “test” on the screen you can use

```
printf("test");
```

but also (the effect is the same)

```
printf(“%s”, "test");
```

Remember that (though it is not explicitly visible) C considers string constants as **including a terminating null character**. For instance, `strcpy(string1, "some text")` copies the characters 's', 'o', 'm', ..., 'x', 't', '\0' into the first 10 elements of array string1.

Usually an array used as a C string is first declared, and only later in the program given the right values to its characters. However, sometimes it is useful to initialize strings just at the moment they are declared. Here are some examples of declaring and initializing C strings at the same time:

```
char string2[20] = { 'h', 'e', 'l', 'l', 'o', '\0' };
/* this is the generic C syntax for array initialization: list of elements
separated by commas, within { }. Note that if you initialize a C string
this way you have to explicitly include character '\0' in the list */

char string3[20] = "hello";
```

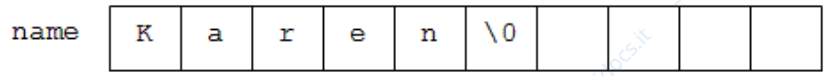
```
/* shortcut array initialization (only allowed for character arrays) using
a string constant; note that, as usual with string constants, character
'\0' is not explicit (but it gets inserted into string3 all the same) */

char string4[20] = "";
/* initialization of string4 as an empty C string of length 0, i.e. a
character array containing a '\0' character as its first element*/
```

Representation in memory

Here is another example of declaring and initializing a C string:

```
char name[10] = "Karen";
```

The following diagram shows how the string `name` is represented in memory:



The individual characters that make up the string are stored in the elements of the array, starting from index 0. The string is terminated by a null character. Array elements after the null character are not part of the string, and their contents are irrelevant when the array is used as a C string. When using array `name`, you should always think of the elements after the null character as having unknown values (and they actually have, until their values have been set by the program).

A "null string", i.e. an empty string, is a C string with a null character as its first character: