Strings

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

Programming II

Definition

- Strings are, actually, one-dimensional arrays of characters (`char`). The only difference with respect to a regular array of characters is that the null character ‘`\0`’ is used to indicate the end of the string (this is not necessary in regular arrays). Because of this, if we intend to hold strings containing a maximum of N characters, we will need a `char` array of size `N+1`.

- For example, to store the word `"hola"`, we could use:

```
char greeting[5]
```

'h'	'o'	'l'	'a'	'\0'

Declaring strings

- To declare a string we can use any of the following three options:

```
char greeting1[] = "hola";
char greeting2[] = {'h','o','l','a','\0'};

char *greeting3 = "hola";
```

Comparing strings: strcmp

- To compare two strings alphabetically (to check if they are equal or to sort them, for example), we use:

```
int strcmp (const char *string1, const char *string2);
```

- This function returns an integer value (`int`) like this:

`0`	if strings are equal
negative value	if `string1 < string2`
positive value	if `string1 > string2`

Modifying a string with the value of another (1): strcpy, strcat

- If we want to set a variable with the value of another one, we use:

```
char *strcpy (char *targetString, const char *sourceString);
```

Modifying a string with the value of another (2): strcpy, **strcat**

- If we want to attach a string at the end of another string (i.e. to concatenate them), we use:

```
char *strcat (char *targetString, const char *sourceString);
```

Reading data: sscanf

- `sscanf` is similar to `scanf`, with the difference that the values of the variables are not read from the keyboard, but from a text string

```
int sscanf (const char *buf, const char *format, ...);
```

- Example:

```
int day, year;
char weekDay[20], month[20], sdate[100];

strcpy(sdate, "Friday January 31 2020");
sscanf(sdate, "%s %s %d %d", weekDay, month, &day, &year);
```

Showing the value of a variable: snprintf

- `snprintf` is similar to `printf`, but writes its output in the string referenced by the first argument (`*str`). In addition, the second argument (`size`) specifies the maximum number of characters that `snprintf` can write in the string, including the termination character ('`\0`').

```
int snprintf (char *str, size_t size, const char *format, ...);
```

- Example:

```
char cadena[20];
char* c = "holamundo!";

snprintf(cadena, 5, "%s\n", c);
printf("Resulting string: %s\n", cadena);
```

```
Resulting string: hola
```