B1 - Unix & C Lab Seminar

B-CPE-101

mini_printf

Bootstrap

1.1

mini_printf

language: C
compilation: gcc *.c

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.

During this bootstrap you will see the notion of **va_args** from the **stdarg.h** headers and how to use it.
You will now use everything you have learn from the C pool.
For this bootstrap and the first project, you will not have to push a main function. You project you will be tested using **gcc *.c**

Don't forget that everything you will create will help you for the next project.

Asking help might help you.

STEP 1: VA_ARGS

Before attempting the next exercices, please take a look at the following man pages and this video.

```
> man va_arg
> man stdarg.h
```

STEP 2: HOW TO USE VA_ARGS

You will implement functions with the following prototypes in their corresponding files:

```
int sum_numbers(int n, ...);
int sum_strings_length(int n, ...);
void disp_stdarg(char *s, ...);
```

You should add those prototypes to the `includes/bsprintf.h` file (do not forget about the include guard in your header).

Don't hesitate to write the tests before adding any features to your program. TDD

Part A: Sum numbers

```
/*
** The sum_numbers() function returns the sum of the numbers passed as parameters
    after n.
** The parameter 'n' represents the number of arguments that will be passed as
    parameter.
** During our tests, the parameter 'n' value will always be lesser or equal to the
    number of parameters given (never greater).
*/
int sum_numbers(int n, ...);
```

Part B: Sum string length

```
/*
** The sum_strings_length() function returns the sum of the lengths of every string
    passed as parameter after n.
** The parameter 'n' represents the number of arguments that will be passed as
    parameter.
** During our tests, the parameter 'n' value will always be lesser or equal to the
    number of parameters given (never greater).
*/
int sum_strings_length(int n, ...);
```

Part C: Display stdargs

```
/*
** This function displays all of its arguments (except the first one),
**  each followed by a '\n', in the order in which they were passed.
** The value of each char composing the first parameter given tells you if the next
** arguement is a char (if the next character is a 'c'), a char* ('s') or an int (an
    'i').
*/
void disp_stdarg(char *s, ...);
```

The whole libC is forbidden, except **write**, **va_start**, **va_arg**, **va_end**.

Unit Tests

It's generally considered a good practice to write unit tests before starting to implement a function. Think about all the cases you should be able to handle!

Here are some basic tests to dispatch to each test files.

```
#include <criterion/criterion.h>
#include <criterion/redirect.h>

Test(sum_numbers, return_sum_numbers)
{
    int ret = sum_numbers(3, 21, 25, -4);
    cr_assert_eq(ret, 42);
}

Test(sum_strings_length, sum_str_lengths) {
    int value = sum_strings_length(5, "Hello", "a", "toto", "", "plop");
    cr_assert_eq(value, 14);
}

Test(disp_stdarg, basic, .init=cr_redirect_stdout) {
    disp_stdarg("csiis", 'X', "hi", 10, -3, "plop");
    cr_assert_stdout_eq_str("X\nhi\n10n-3\nplop\n", "");
}
```