B1 - Phoenix Bootcamp

B-BOO-101

Day 02

Recursive and strings

Day 02

repository name: BOO_phoenix_d02_*$ACADEMICYEAR*
repository rights: ramassage-tek
language: C
compilation: gcc *.c (+ our own files containing the `main` function)

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

Don't push your `main` and `my_putchar` functions into your delivery directory. Your files will be compiled adding our own `main` and `my_putchar`.

You are only allowed to use the `my_putchar` function to complete the following tasks, but don't push it into your delivery directory, and don't copy it in *any* of your delivered files.

Task 01 - iterative_factorial

Delivery: `iterative_factorial.c`

Write an iterative function that returns the factorial of the number given as a parameter.
It must be prototyped the following way:

```c
int iterative_factorial(int nb);
```

In case of error, the function should return 0.

0! = 1
if n < 0, n! = 0

Task 02 - recursive_power

Delivery: `recursive_power.c`

Write a recursive function that returns the first argument raised to the power `p`, where `p` is the second argument.
It must be prototyped the following way:

```c
int recursive_power(int nb, int p);
```

In case of error, the function should return 0.

n^0 = 1
if p < 0, n^p = 0

Task 03 - is_prime_number

Delivery: `is_prime_number.c`

Write a function that returns `1` if the number is prime and `0` if not.

```c
int is_prime_number(int nb);
```

As you know, `0` and `1` are not prime numbers.

From here on, most of the tasks functions exist in the **string** library. Use `man` to obtain a full explanation of how a function works. Beware that none of your deliveries contains a function frmo this **string** library!

Task 04 - my_strcpy

Delivery: `my_strcpy.c`

Write a function that copies a string into another. The destination string will already have enough memory to copy the source string.
It must be prototyped the following way:

```
char *my_strcpy(char *dest, char const *src)
```

The function returns `dest`.

Task 05 - my_strncmp

Delivery: `my_strncmp.c`

Reproduce the behavior of the `strncmp` function. Your function must be prototyped the following way:

```
int my_strncmp(char const *s1, char const *s2, int n);
```

TASK 06 - MY_STRSTR

Delivery: `my_strstr.c`

Reproduce the behavior of the `strstr` function. Your function must be prototyped the following way:

```
char *my_strstr(char *str, char const *to_find);
```

TASK 07 - MY_STRUPCASE

Delivery: `my_strupcase.c`

Write a function that puts every letter of every word in it in uppercase.
It must be prototyped the following way:

```
char *my_strupcase(char *str);
```

The function returns `str`.

TASK 08 - MY_STRCAPITALIZE

Delivery: `my_strcapitalize.c`

Write a function that capitalizes the first letter of each word.
It must be prototyped the following way:

```
char *my_strcapitalize(char *str);
```

The function returns `str`.

The sentence, `hey, how are you? 42WORds forty-two; fifty+one`
will become `Hey, How Are You? 42words Forty-Two; Fifty+One`

TASK 09 - UNIT TESTS

Delivery: tests/test_my_strcpy.c, tests/test_my_strncmp.c

You have to write unit tests (using Criterion) for some tasks: my_strcpy and my_strncmp.
You **MUST** have at least a line coverage of 60% and a branch coverage of 40% for both of these functions.
Here's an example of unit test for the function my_strcpy:

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

char *my_strcpy(char *dest, char const *src);

Test(my_strcpy, copy_in_empty_array)
{
    char dest[6] = {0};

    my_strcpy(dest, "Hello");
    cr_assert_str_eq(dest, "Hello");
}
```

You can even compare the result with the libC functions like so:

```c
Test(my_strcpy, copy_string_in_empty_array)
{
        char my_dest[6] = {0};
        char dest[6] = {0};

        my_strcpy(my_dest, "Hello");
        strcpy(dest, "Hello");
        cr_assert_str_eq(my_dest, dest);
}
```

Take a look at the unit tests documentation.
Your test will be built manually (without Makefile) using the command shown in the documentation.