B1- Unix and C Lab Seminar

B-CPE-100

Day 06

Pointers are back

v2.0

{EPITECH.}

Day 06

Pointers are back

repository name:	CPool_Day06_$ACADEMICYEAR
repository rights:	ramassage-tek
language:	C
group size:	1

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- Don't push your **main** function into your delivery directory, we will be adding our own. Your files will be compiled adding our **main.c** and our **my_putchar.c** files.
- 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.
- If one of your files prevents you from compiling with * **.c**, the Autograder will not be able to correct your work and you will receive a 0.

Create your repository at the beginning of the day and submit your work on a regular basis!
The delivery directory is specified within the instructions for each task.
In order to keep your repository clean, pay attention to `gitignore`.

Most of the day's 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 from this **string** library!

Introduction

Unit Tests

It is highly recommended to test your functions as you develop them. It is common practice to create and use what is called **unit tests**.
Today, each task will be splitt in two parts: create the function in one part and test it in another part.
To help you write good unit tests in C, you will use the Criterion framework. To do that, please follow the instructions on the document "How to write Unit Tests" from the intranet, available here.

You don't know Makefile yet, so your test will be built manually using the command shown in the documentation.

Task 01

my_strcpy

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`.

Delivery: my_strcpy.c

Task 01.5

test_my_strcpy

As ask in the introduction, you have to write unit tests (using Criterion) for all of today's task. Starting now with the my_strcpy function.

Here's an example of two unit tests for this function:

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

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

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

Test(my_strcpy, copy_string_in_empty_array_return_value)
{
        char dest[6] = {0};
        char *copy = my_strcpy(dest, "Hello");

        cr_assert_str_eq(copy, "Hello");
}
```

Delivery: tests/test_my_strcpy.c

Task 02

my_strncpy

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

```
char *my_strncpy(char *dest, char const *src, int n);
```

The function returns `dest`.
Delivery: my_strncpy.c

Add '\0's if n is strictly greater than the length of the string. Do not add '\0' if n is strictly lower than the length of the string (because `dest` is not supposed to contain more than n bytes.

Task 02.5

test_my_strncpy

We continue the unit testing with the my_strncpy function.

Here's an example of unit test for this function:

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

Test(my_strncpy, copy_five_characters_in_empty_array)
{
        char dest[6] = {0};

        my_strncpy(dest, "HelloWorld", 5);
        cr_assert_str_eq(dest, "Hello");
}
```

Delivery: tests/test_my_strncpy.c

Task 03

my_revstr

Write a function that reverses a string. It must be prototyped the following way:

```
char *my_revstr(char *str);
```

The function returns `str`.
Delivery: my_revstr.c

Task 03.5

test_my_revstr

We continue the unit testing with the my_revstr function.

Here's an example of unit test for this function:

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

Test(my_revstr, return_value_is_reversed)
{
        cr_assert_str_eq(my_revstr("Hello"), "olleH");
}
```

Delivery: tests/test_my_revstr.c

You've now understand the principle for the rest of the day. You have to test **ALL** of today's functions (except specified otherwise).

Task 04

my_strstr

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

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

Delivery: my_strstr.c, tests/test_my_strstr.c

Check out the *my_strcmp* and *my_strncmp* functions.

Task 05

my_strcmp

Reproduce the behavior of the *strcmp* function. Your function must be prototyped the following way:

```
int my_strcmp(char const *s1, char const *s2);
```

Delivery: my_strcmp.c, tests/test_my_strcmp.c

Task 06

my_strncmp

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);
```

The function should return the same values as *strcmp(3)*.
Delivery: my_strncmp.c, tests/test_my_strncmp.c

Task 07

my_strupcase

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*.
Delivery: my_strupcase.c, tests/test_my_strupcase.c

Task 08

my_strlowcase

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

```
char *my_strlowcase(char *str);
```

The function returns *str*.
Delivery: my_strlowcase.c, tests/test_my_lowcase.c

Don't forget to write unit tests for each functions!

Task 09

my_strcapitalize

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*.
Delivery: my_strcapitalize.c, tests/test_my_strcapitalize.c

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

Task 10

my_str_isalpha

Write a function that returns **1** if the string passed as parameter only contains alphabetical characters and **0** if the string contains another type of character. It must be prototyped the following way:

```
int my_str_isalpha(char const *str);
```

The function returns **1** if *str* is an empty string.
Delivery: my_str_isalpha.c, tests/test_my_str_isalpha.c

Task 11

my_str_isnum

Write a function that returns **1** if the string passed as parameter only contains digits and **0** otherwise.
It must be prototyped the following way:

```
int my_str_isnum(char const *str);
```

The function returns **1** if *str* is an empty string.
Delivery: my_str_isnum.c, tests/test_my_str_isnum.c

Task 12

my_str_islower

Write a function that returns **1** if the string passed as parameter only contains lowercase alphabetical characters and **0** otherwise.
It must be prototyped the following way:

```
int my_str_islower(char const *str);
```

The function returns **1** if the *str* is an empty string.
Delivery: my_str_islower.c, tests/test_my_str_islower.c

Task 13

my_str_isupper

Write a function that returns **1** if the string passed as parameter only contains uppercase alphabetical characters and **0** otherwise.
It must be prototyped the following way:

```
int my_str_isupper(char const *str);
```

The function returns **1** if *str* is an empty string.
Delivery: my_str_isupper.c, tests/test_my_str_isupper.c

Task 14

my_str_isprintable

Write a function that returns **1** if the string passed as parameter only contains printable characters and **0** otherwise.
It must be prototyped the following way:

```
int my_str_isprintable(char const *str);
```

The function returns **1** if *str* is an empty string.
Delivery: my_str_isprintable.c, tests/test_my_str_isprintable.c

man isprint

Again, don't forget to write unit tests for each functions! It will help you ensure that they work as intended.

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

```
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);
}
```

For the remaining functions, we don't ask you to write unit tests, but we still encourage you to do so. Look at the redirect.c sample of Criterion to learn how to test things written on standard output.

Task 15

my_putnbr_base

Write a function that converts and displays a decimal number into a number in a given base.
The number is given as an *int* and the base is provided as a *string*.
The base contains all the symbols that can be used to print a number (for instance, *0123456789* for the decimal base, *01* for the binary base, *0123456789ABCDEF* for the hexadecimal base).
The function must deal with negative numbers, and be prototyped the following way:

```
int my_putnbr_base(int nbr, char const *base);
```

Delivery: my_putnbr_base.c

Task 16

my_getnbr_base

Write a function that converts and returns a number (provided as a *string*) in a given base into a decimal number.
The function must deal with negative numbers, and several successive **+** or **-** before the number.
If any error occurs, the function must return **0**. It must be prototyped the following way:

```
int my_getnbr_base(char const *str, char const *base);
```

Delivery: my_getnbr_base.c

Task 17

my_showstr

Write a function that prints a string and returns **0**. If this string contains non-printable characters, they must be printed hexadecimally (in lowercase letters) with a backslash before the given value.
It must be prototyped the following way:

```
int my_showstr(char const *str);
```

Delivery: my_showstr.c

For instance, *I like \n ponies!* will be printed as *I like \0a ponies!*.

Task 18

my_showmem

Write a function that prints a memory dump and return **0**. It must be prototyped the following way:

```
int my_showmem(char const *str, int size);
```

Each line of the output manages 16 characters and is divided into three columns:

- The hexadecimal address of the line's first character,
- the content in hexadecimal,
- the content in printable characters.

Any non printable characters must be replaced by a dot.

Terminal

```
~/B-CPE-100> ./my_showmem | cat -e
00000000:  6865 7920 6775 7973 2073 686f 7720 6d65 hey guys show me$
00000010:  6d20 6973 2063 6f6f 6c20 796f 7520 6361 m is cool you ca$
00000020:  6e20 646f 2073 6f6d 6520 7072 6574 7479 n do some pretty$
00000030:  206e 6561 7420 7374 7566 6600 0f1b 7f05  neat stuff.....$
00000040:  2e00 0102 0304 0506 0708 090e 0f1b 7f   ..............$
```

Delivery: my_showmem.c

Don't forget the padding if there aren't enough characters to have a valid alignment.