B1 - Unix & C Lab Seminar

B-CPE-100

Day 09

Structures

Day 09

language: C

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

- 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`.
- 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.

All .c files from your delivery folder will be collected and compiled with your `libmy`, which must be found in `lib/my/`. For those of you using .h files, they must be located in `include/` (like the `my.h` file).

Some tests will automatically compile your functions the following way:

Terminal

```
~/B-CPE-100> cd taskXX
~/B-CPE-100> gcc *.c -c -I../include/
~/B-CPE-100> gcc *.o autograder/main_taskXX.o -L../lib/my/ -o taskXX -lmy
```

Your library will be built using the `lib/my/build.sh` script you previously made (see Day07).

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`.

Allowed system function(s): `write`, `malloc`, `free`

We still encourage you to write unit tests for all your functions!
Check out Day06 if you need an example, and re-read the guide.

TASK 01 - MY_MACRO_ABS.H

Delivery: include/my_macro_abs.h

Write a macro, named `ABS`, that replaces an argument with an absolute value:

```
#define ABS(value)
```

TASK 02 - MY.H

Delivery: include/my.h

Write your `my.h` header file that contains the prototypes of all the functions exposed by your `libmy.a`.

To check exposed functions, see the man of `nm`.

Have you heard about **static functions**?

TASK 03 - MY_PARAMS_TO_ARRAY

Delivery: my_params_to_array.c

Write a function that stores the program's parameters into an array of structures and returns the address of the array's first cell. All array elements are to be addressed, including `av[0]`.
The function must be prototyped as follows:

```
struct info_param *my_params_to_array(int ac, char **av);
```

The structures contained is the array are to be allocated.
To indicate the end of the array, the `str` field of its last cell must be set to `0`.
The structure is defined as follows:

```
struct info_param
{
    int length;         // parameter's length
    char *str;          // parameter's address
    char *copy;         // parameter's copy
    char **word_array; // the result of my_str_to_word_array(str)
};
```

Do not submit the `struct info_param` structure; the tests set will use its own.

Your function will be tested with your own `my_show_word_array`.
As we will not compile `my_show_word_array.c`, you need to make it work using your library.

TASK 04 - MY_SHOW_PARAM_ARRAY

Delivery: my_show_param_array.c

Write a function that displays the content of an array created with the previous function, and prototyped as follows:

```
int my_show_param_array(struct info_param const *par);
```

Do not submit the `struct info_param` structure; the tests set will use its own.
For each cell, display one of the following elements per line: parameter, size and words (one per line).

Your function will be tested with your own `my_str_to_word_array`.
As we will not compile `my_str_to_word_array.c`, you need to make it work using your library.

TASK 05 - GET_COLOR

Delivery: get_color.c

Write a function that returns the color as an `int` by handling its three *RGB* components.
The function must be prototyped as follows:

```
int get_color(unsigned char red, unsigned char green, unsigned char blue);
```

This task is *only* to be completed with **bit shifts**.

TASK 06 - SWAP_ENDIAN_COLOR

Delivery: swap_endian_color.c

Write a function that changes the endianness of the color and returns it.
The color should be ordered like this: ARGB
The function must be prototyped as follows:

```
int swap_endian_color(int color);
```

This task has to be completed with a `union`.

You will only be working with big and little endians.