DAY 11

LINKED LISTS

```
$> gcc main.c
$> ./a.out
Critical Success!
...
Hello World of Tech
```

DAY 11

Preliminaries

Language: C

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

Your `libmy.a` must have a Makefile in order to be built!

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.

For the tasks regarding linked lists, we will be using the following structure:

```
typedef struct linked_list {
        void *data;
        struct linked_list *next;
} linked_list_t;
```

This structure **must be found** in a file named `mylist.h` in your includes folder.

Task 01 - my_params_to_list

Delivery: my_params_to_list.c

Write a function named `my_params_to_list` that creates a new list from the command line arguments. The address of the list's first node is returned.
It must be prototyped as follows:

```
linked_list_t *my_params_to_list(int ac, char * const *av);
```

```
Terminal
~/B-CPE-100> ./a.out test arg2 arg3
```

If the main function directly transmits its `argc`/`argv` arguments to `my_params_to_list`, the function must place `./a.out` first on the list, then `test`, `arg2` and `arg3`.
When scanning the list, we will have `arg3` as the first element, then `arg2`, ... and finally, `./a.out`.

Task 02 - my_list_size

Delivery: my_list_size.c

Write a function called `my_list_size` that returns the number of elements on the list.
It must be prototyped as follows:

```
int my_list_size(linked_list_t const *begin);
```

Task 03 - my_rev_list

Delivery: my_rev_list.c

Write a function named `my_rev_list` that reverses the order of the list's elements.
It should be prototyped as follows:

```
void my_rev_list(linked_list_t **begin);
```

Task 04 - my_apply_on_nodes

Delivery: my_apply_on_nodes.c

Write a function named `my_apply_on_nodes` that applies a function, given as argument, to the data of each node on the list.
It must be prototyped as follows:

```
int my_apply_on_nodes(linked_list_t *begin, int (*f)(void *));
```

The function pointed by `f` will be used as follows: `(*f)(list_ptr->data);`

Task 05 - my_apply_on_matching_nodes

Delivery: my_apply_on_matching_nodes.c

Write a function named `my_apply_on_matching_nodes` that applies a function, given as argument, to the data of the nodes on the list equal to the data_ref given as argument.
The function must be prototyped as follows:

```
int my_apply_on_matching_nodes(linked_list_t *begin, int (*f)(), void const *data_ref, int
     (*cmp)());
```

The functions pointed by `f` and `cmp` will be used as follows: `(*f)(list_ptr->data);` and `(*cmp)(list_ptr->data, data_ref);`

The `cmp` function could be `my_strcmp`; the elements are only considered equal if `cmp` returns 0 (data is *equal*).

Task 06 - my_find_node

Delivery: my_find_node.c

Write a function named `my_find_node` that returns the address of the first node, which contains data *equal* to the reference data.
It must be prototyped as follows:

```
linked_list_t *my_find_node(linked_list_t const *begin, void const *data_ref, int (*cmp)()
    );
```

Task 07 - my_delete_nodes

Delivery: my_delete_nodes.c

Write a function named `my_delete_nodes` that removes all nodes containing data *equal* to the reference data.
It must be prototyped as follows:

```
int my_delete_nodes(linked_list_t **begin, void const *data_ref, int (*cmp)());
```

Task 08 - my_concat_list

Delivery: my_concat_list.c

Write a function named `my_concat_list` that puts the elements of a `begin2` list at the end of a `begin1` list.
It must be prototyped as follows:

```
void my_concat_list(linked_list_t **begin1, linked_list_t *begin2);
```

Creating elements is not allowed! You must link the two lists together.

Task 09 - my_sort_list

Delivery: my_sort_list.c

Write a function named `my_sort_list` that sorts a list in ascending order by comparing data, node-to-node, with a comparison function.
It must be prototyped as follows:

```c
void my_sort_list(linked_list_t **begin, int (*cmp)());
```

Task 10 - my_add_in_sorted_list

Delivery: my_add_in_sorted_list.c

Write a function named `my_add_in_sorted_list` that creates a new element and inserts it into an sorted list, so that the list remains sorted in ascending order.
It must be prototyped as follows:

```c
void my_add_in_sorted_list(linked_list_t **begin, void *data, int (*cmp)());
```

Task 11 - my_merge

Delivery: my_merge.c

Write a function named `my_merge` that integrates the elements of a sorted list, `begin2`, into another sorted list, `begin1`, so that `begin1` remains sorted in ascending order.
It must be prototyped as follows:

```c
void my_merge(linked_list_t **begin1, linked_list_t *begin2, int (*cmp)());
```

Watch out for `NULL` pointers!

v 2.1

{EPITECH}