
# B1- Unix and C Lab Seminar

B-CPE-100

# Day 11

Linked lists

v2.1

{EPITECH.}

# Day 11

## Linked lists

| | |
|---|---|
| **repository name:** | CPool_Day11_$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**.
- 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 is found in **CPool_Day11_$ACADEMICYEAR/lib/my**. For those of you using **.h** files, they must be located in **CPool_Day11_$ACADEMICYEAR/include**.

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

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.

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

We still encourage you to write unit tests for all your functions!

# Task 01

## my_params_to_list

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);
```

**Delivery:** CPool_Day11_$ACADEMICYEAR/my_params_to_list.c
For instance,

```
Terminal
~/B-CPE-100> ~/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

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);
```

**Delivery:** CPool_Day11_$ACADEMICYEAR/my_list_size.c

# Task 03

## my_rev_list

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);
```

**Delivery:** CPool_Day11_$ACADEMICYEAR/my_rev_list.c

# Task 04

## my_apply_on_nodes

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 *));
```

**Delivery:** CPool_Day11_$ACADEMICYEAR/my_apply_on_nodes.c

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

# Task 05

## my_apply_on_matching_nodes

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)());
```

**Delivery:** CPool_Day11_$ACADEMICYEAR/my_apply_on_matching_nodes.c

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

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)());
```

**Delivery:** CPool_Day11_$ACADEMICYEAR/my_find_node.c

# Task 07

## my_delete_nodes

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)());
```

**Delivery:** CPool_Day11_$ACADEMICYEAR/my_delete_nodes.c

# Task 08

## my_concat_list

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);
```

**Delivery:** CPool_Day11_$ACADEMICYEAR/my_concat_list.c

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

# Task 09

## my_sort_list

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:

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

**Delivery:** CPool_Day11_$ACADEMICYEAR/my_sort_list.c

# Task 10

## my_add_in_sorted_list

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:

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

**Delivery:** CPool_Day11_$ACADEMICYEAR/my_add_in_sorted_list.c

# Task 11

## my_merge

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:

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

**Delivery:** CPool_Day11_$ACADEMICYEAR/my_merge.c

Watch out for **NULL** pointers!