C - Pool - Tek1

Subject Day 11

C Pool Managers
looneytunes@epitech.eu

Contents

Instructions

- The subject may change up to one hour before turn in.
- Respect the norm takes time, but is good for you. This way your code will respect the norm since the first written line.
- Ask yourself if it's relevant to let a `main()` function in your turn-in knowing we will add our own.
- For each exercise's turn in folder, we will compile your files using the `cc -c *.c`, command, thus generating `.o` files that we will then link one by one, adding our own `main.c`:

```
$> cd ex_01
$> cc *.c -c -I../include/
$> cc *.o ~moulinette/main_ex_01.o -L../lib/ -o ex01 -lmy
$> ./ex01
[...]
```

- This is a turn-in directory, of course you will only keep in it your final work revision. No temporary file should stand there!
  You must not leave any file in your turn in folder apart from those explicitly mentioned in the exercises.
  If one of your files prevents from compiling with `*.c`, the robot will not be able to correct your work and you will get a `0`. You should rather delete the exercises that do not work.
- You must have no `my_putchar` function in the files you turn in, it is already in your library.
- You can discuss about it in the pool section of the forum !
- Turn in folder:
  `Piscine_C_J11`

 *Hints* `Remember it is always better to create your repository at the beginning of the day and to turn-in your work on a regular basis`

 *Hints* `On the instructions of each exercises, this directory is specified for every turn-in path`

- Your library will be used during linking phase.
- For the exercises about lists, we will use the following structure (slightly different from the one seen in the lesson):

```
1   typedef struct s_list
2   {
3     void *data;
4     struct s_list *next;
5   } t_list;
```

- That structure must be found in a file named `mylist.h` in your includes folder.

Unit Tests

- It is highly recommended to test your functions when you are developing them.
- Usually, it is common to create a function named "`main`" (and a dedicated file to host it) to check the functions separately.
- Create a directory named "`tests`".
- Create a function "`int main()`" in a file named "`tests-exercise_name.c`", stored inside the directory "`tests`" previously created.
- According to you, this function must contains all the necessary call to "`exercise_name`" to cover all possible cases (special or regular) of the function.

Exercise 1 - my_params_in_list

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

```
1   t_list *my_params_in_list(int ac, char **av);
```

- Example:

```
$> ./a.out test arg2 arg3
```

  - If the `main` function transmits directly its arguments (`argc` and `argv`) to `my_params_in_list` it shall put `./a.out` first in the list, then `test`, `arg2` and `arg3`. When browsing the list, we will have `arg3` as first element, then `arg2`, `test` and `./a.out`.

- Turn in folder:
`Piscine_C_J11/ex_01/my_params_in_list.c`

Exercise 2 - my_list_size

- Write a function named `my_list_size` that returns the number of elements in the list.
- It shall be prototyped as follows:

```
1   int my_list_size(t_list *begin);
```

- Turn in folder:
`Piscine_C_J11/ex_02/my_list_size.c`

Exercise 3 - my_rev_list

- Write a function named `my_rev_list` that reverses the order of the elements in the list. You are only allowed to manipulate pointers.
- It shall be prototyped as follows:

```
1  int my_rev_list(t_list **begin);
```

- Turn in folder:
  `Piscine_C_J11/ex_03/my_rev_list.c`

Exercise 4 - my_apply_on_list

- Write a function named `my_apply_on_list` that applies a function given as argument to the data of each node in the list.
- It shall be prototyped as follows:

```
1   int my_apply_on_list(t_list *begin, int (*f)(void*));
```

- The function pointed by `f` will be used as follows:

```
1   (*f)(list_ptr->data);
```

- Turn in folder:
`Piscine_C_J11/ex_04/my_apply_on_list.c`

Exercise 5 - my_apply_elm_eq_in_list

- Write a function named `my_apply_on_eq_in_list` that applies a function given as argument to the data of certain nodes in the list. A reference information and a comparison function will allow us to select the proper nodes: those "equal" to the reference information.
- It shall be prototyped as follows:

```
1   int my_apply_on_eq_in_list(t_list *begin, int (*f)(), void *data_ref, int (*cmp)());
```

- The functions pointed by `f` and `cmp` will be used as follows:

```
1   (*f)(list_ptr->data);
2   (*cmp)(list_ptr->data, data_ref);
```

- Turn in folder:
  `Piscine_C_J11/ex_05/my_apply_on_eq_in_list.c`

Hints The cmp function could be my_strcmp, the elements are considered equal only if cmp returns 0 (data are "equal").

Exercise 6 - my_find_elm_eq_in_list

- Write a function named `my_find_elm_eq_in_list` that returns the data of the first node "equal" to the reference data.
- It shall be prototyped as follows:

```
1   void *my_find_elm_eq_in_list(t_list *begin, void *data_ref, int (*cmp)());
```

- Turn in folder:
`Piscine_C_J11/ex_06/my_find_elm_eq_in_list.c`

Hints `The cmp function could be my_strcmp, the elements are considered equal only if cmp returns 0 (data are "equal").`

Exercise 7 - my_find_node_eq_in_list

- Write a function named `my_find_node_eq_in_list` that returns the address of the first node containing data "equal" to the reference data.
- It shall be prototyped as follows:

```
1   t_list *my_find_node_eq_in_list(t_list *begin, void *data_ref, int (*cmp)());
```

- Turn in folder:
`Piscine_C_J11/ex_07/my_find_node_eq_in_list.c`

Hints The cmp function could be my_strcmp, the elements are considered equal only if cmp returns 0 (data are "equal").

Exercise 8 - my_rm_all_eq_from_list

- Write a function named `my_rm_all_eq_from_list` that erases all elements containing data "equal" to the reference data.
- It shall be prototyped as follows:

```
1   int my_rm_all_eq_from_list(t_list **begin, void *data_ref, int (*cmp)());
```

- Turn in folder:
`Piscine_C_J11/ex_08/my_rm_all_eq_from_list.c`

Hints `The cmp function could be my_strcmp, the elements are considered equal only if cmp returns 0 (data are "equal").`

Exercise 9 - my_add_list_to_list

- Write a function named `my_add_list_to_list` that puts the elements of a list `begin2` at the end of another list `begin1`.
- Creating elements is not allowed.
- It shall be prototyped as follows:

```
1   int my_add_list_to_list(t_list **begin1, t_list *begin2);
```

- Turn in folder:
`Piscine_C_J11/ex_09/my_add_list_to_list.c`

Exercise 10 - my_sort_list

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

```
1   int my_sort_list(t_list **begin, int (*cmp)());
```

- Turn in folder:
`Piscine_C_J11/ex_10/my_sort_list.c`

Hints `The cmp function could be my_strcmp`

- That is:
  - if `cmp` returns `0`, data are “equal”
  - if `cmp` returns a negative value, the first data is lesser than the second
  - if `cmp` returns a positive value, the first data is greater than the second

Exercise 11 - my_put_elem_in_sort_list

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

```
1   int my_put_elem_in_sort_list(t_list **begin, void *data, int (*cmp)());
```

- Turn in folder:
`Piscine_C_J11/ex_11/my_put_elem_in_sort_list.c`

Hints `The cmp function could be my_strcmp`

- That is:
  - if `cmp` returns `0`, data are “equal”
  - if `cmp` returns a negative value, the first data is lesser than the second
  - if `cmp` returns a positive value, the first data is greater than the second

Exercise 12 - my_add_sort_list_to_sort_list

- Write a function named `my_add_sort_list_to_sort_list` that integrates the elements of an ordered list `begin2` in another ordered list `begin1`, so that the `begin1` list remains sorted in ascending order.
- It shall be prototyped as follows:

```
1   int my_add_sort_list_to_sort_list(t_list **begin1, t_list *begin2, int (*cmp)());
```

- Turn in folder:
`Piscine_C_J11/ex_12/my_add_sort_list_to_sort_list.c`

Hints `The cmp function could be my_strcmp`

- That is:
  - if `cmp` returns `0`, data are “equal”
  - if `cmp` returns a negative value, the first data is lesser than the second
  - if `cmp` returns a positive value, the first data is greater than the second

`Pay attention to NULL pointers!`