# C - Pool - Tek1

## Subject Day 04

C Pool Managers
looneytunes@epitech.eu

# Contents

# Instructions

- The subject may change until 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.
- We will compile your files with the command `cc *.c`, adding our `main.c` and our `my_putchar.c` :

```
$> cc *.c ~moulinette/main_ex_01.c ~moulinette/my_putchar.c -o ex01
$> ./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 shall leave in your directory no other files than those explicitly specified by the exercises.
  If one of your files prevents the compilation with `*.c`, the robot will not be able to do the correction and you will have a `0`. That is why it's in your interest to remove any file that doesn't work.

 *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`

- You are only allowed to use the `my_putchar` function to do the following exercises. This function will be provided, so:
  - You shall not have a `my_putchar.c` file in your turn-in directory.
  - The function `my_putchar` shall not be in any of your turned-in files.
- Don't forget to discuss about it in the pool section of the forum !

# 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.

*Indices*

```
Here is a partial list of tests:
- Check the empty strings
- Pay attention to the possible values of an int
(0, min and max)
```



# Exercise 1 - Machine Exam

- Open your weekly schedule in your intranet.

- Enroll in saturday's exam.
  => If you don't know how to do it, go back to step one.

- Check you're enrolled in saturday's exam.
  => If you are not, return to previous step.

- Check that you checked you're enrolled.
  => If you didn't check, return to previous step.

- Are you enrolled in saturday's exam ?
  => If you are not, return to step one.



# Exercise 2 - my_swap

- Write a function that swaps the contents of two integers whose adresses are given as parameters

- It will be prototyped as follows:

```
1   int my_swap(int *a, int *b);
```

- Turn-in directory:
`Piscine_C_J04/my_swap.c`

# Exercise 3 - my_putstr

- Write a function that displays on the screen, one by one, the characters of a string.
- The address of the first character of the string is contained in the pointer passed as a parameter to the function.
- It will be prototyped as follows:

```
1   int my_putstr(char *str);
```

- Turn-in directory:
`Piscine_C_J04/my_putstr.c`

# Exercise 4 - my_strlen

- Write a function that counts and returns the number of characters found in the string passed as a parameter to the function.
- It will be prototyped as follows:

```
1   int my_strlen(char *str);
```

- Turn-in directory:
`Piscine_C_J04/my_strlen.c`

# Exercise 5 - my_evil_str

- The goal of this exercise is to swap each characters of the 'str' string two by two. Thus, we swap the first letter with the last one, the second with the penultimate, and so on.
- This function will return a pointer to the first character of the reversed string.

```
1    char *my_evil_str(char *str);
```

- Turn-in directory:
`Piscine_C_J04/my_evil_str.c`
- Examples:

```
1    a => a
2    ab => ba
3    abc => cba
4    abcd => dcba
5    abcde => edcba
6    abcdef => fedcba
7    ...
```

- Warning, with the code below, the string 'str' will be ReadOnly:

```
1 char *str;
2
3 str = "toto";
```

- If you try to modify this string you will have a segfault.
- To be able to test your program, you will need to duplicate the string in ReadWrite mode using: `strdup()`.
This is an example of a main:

```
1  #include <string.h>
2
3  int main()
4  {
5   char *str;
6
7   str = strdup("toto");
8   my_evil_str(str);
9   my_putstr(str);
10  return (0);
11 }
```

 *Hints* For the curious, you are going to code your own strdup in a few days, be patient !

# Exercise 6 - my_getnbr

- Write a function that returns a number. This number will be given to the function as a string (Ex: "842") and needs to be returned as an integer.
- We consider that the number will be in base ten. ("0123456789").
- It will be prototyped as follows:

```
1    int my_getnbr(char *str);
```

- Your function will manage those strings:

"+---+--++---+---+---+-42" returns -42
"42a43" returns 42
"11000000000000000000000042" returns 0 because this number doesn't fit in an integer.
"-10000000000000000000000042" returns 0 for the same reason

- Turn-in directory:
`Piscine_C_J04/my_getnbr.c`

# Exercise 7 - my_sort_int_tab (Bonus)

- Write a function that sort a table made of integers in ascending order.
- The parameters are: a pointer to integer and the number of integers in the table.

```
1   void my_sort_int_tab(int *tab, int size);
```

- Turn-in directory:
  `Piscine_C_J04/my_sort_int_tab.c`