# C - Pool - Tek1

## Subject Day 09

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`:
- For those who use .h files, they must be found in:
  `Piscine_C_J09/include`
- The robot will test your turn-in as follows:

```
$> cd ex\_01
$> cc -c *.c -I../include/
$> cc *.o ~moulinette/main_ex_01.o -L../lib/my/ -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 exercises that do not work.
- You can discuss about it in the pool section of the forum !
- Turn in folder:
  `Piscine_C_J09`

 *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`

- We compile with your lib and your includes. So, they must be put respectively into: `Piscine_C_J09/lib/my/`, `Piscine_C_J09/include/`
- The robot uses your library. If your `my_str_to_wordtab` or `my_show_wordtab` does not work, do not try exercises 3 and 4.

# 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
- Check the pointer's values
```

# Exercise 1 - my_macroABS.h

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

```
1   #define ABS(Value)
```

- Turn in folder:
`Piscine_C_J09/ex_01/my_macroABS.h`

# Exercise 2 - my.h

- Write your `my.h` header file.
- It contains the prototypes of all functions in your `libmy.a`.
- Turn in folder:
  `Piscine_C_J09/include/my.h`

# Exercise 3 - my_param_to_tab

- Write a function that stores the program's parameters in an array of structures and returns the address of the first cell of the array.
- All array elements shall be addressed, including `av[0]`.
- It shall be prototyped as follows:

```
1   struct s_stock_par *my_param_to_tab(int ac, char **av);
```

- The array of structure shall be allocated, and the last cell shall contain 0 in its `str` element to indicate the end.
- The structure is defined as follows:

```
1   struct s_stock_par
2   {
3     int size_param;
4     char *str;
5     char *copy;
6     char **tab;
7   };
```

  - `size_param` being the length of the parameter
  - `str` being the address of the parameter
  - `copy` being the copy of the parameter
  - `tab` being the array returned by `my_str_to_wordtab`

- You must not turn in the `struct s_stock_par` structure, the robot will use its own, along with the following typedef:

```
1   typedef struct s_stock_par t_stock_par;
```

- We test your function with `my_show_wordtab`. Make it work (we do not compile `my_show_wordtab.c`).
- Turn in folder:

`Piscine_C_J09/ex_03/my_param_to_tab.c`

# Exercise 4 - my_show_tab

- Write a function that displays the content of an array created with the previous function.
- It shall be prototyped as follows:

```
1   int my_show_tab(struct s_stock_par *par);
```

- You must not turn in the `struct s_stock_par` structure, the robot will use its own, along with the following typedef:

```
1   typedef struct s_stock_par t_stock_par;
```

- For each cell, display (one element per line):
  - the parameter
  - the size
  - each word (one per line)
- We test your function with `my_show_wordtab`. Make it work (we do not compile `my_show_wordtab.c`).
- Turn in folder:
  `Piscine_C_J09/ex_04/my_show_tab.c`

# Exercise 5 - get_color

- Write a fonction which returns the color as an int by handling its three components RGB.
- The color will be ordered like: ARGB.
- It shall be prototyped as follows:

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

- Turn in folder:
`Piscine_C_J09/ex_05/get_color.c`

This exercise has to be done only with bit shifts.

# Exercise 6 - swap_endian_color

- Write a function that change the endianness of the color and return it.
- The color will be ordered as: ARGB.
- It shall be prototyped as follows:

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

- Turn in folder:
`Piscine_C_J09/ex_06/swap_endian_color.c`

*Indices* `You have to handle only big endian and little endian.`

`This exercise has to be done with an union.`