C - Pool - Tek1

Subject Day 10

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.
- We will compile with your library and your includes.
- Turn-in directory:
  `Piscine_C_J10`

 *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`

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 `The tests are not applicable on binaries.`

Exercise 1 - Makefile

- Write your `Makefile` which compiles your `libmy`.
- The `Makefile` should copy the `library` into :
  `Piscine_C_J10/lib`
- The `Makefile` should copy the `my.h` into :
  `Piscine_C_J10/include`
- The `Makefile` should implement the rule `clean`.
- Your `Makefile` and all your other files (necessary `.c` and `.h`), will be in:
  `Piscine_C_J10/lib/my/`

Exercise 2 - do-op

- Write a program called `do-op`.
- The program must be executed with three arguments:
  `do-op value1 operator value2`
- Exemple :

```
$>./do-op 42 "+" 21
63
$>
```

- The character `operator` will correspond to the appropriate function into an array of function pointers.
- This directory must have a `Makefile` with the rules `all, clean, fclean, re` and must not relink.
- If the expression is wrong `./do-op foo devide bar` the program must display `0`.
- If the number of arguments is not correct `do-op` must display nothing.
- Some of the tests from the grading robot:

```
$> make clean
$> make
$> ./do-op
$> ./do-op 1 + 1
2
$> ./do-op 42friends - ---20toto12
62
$> ./do-op 1 p 1
0
$> ./do-op 1 +toto 1
2
$> ./do-op 1 + toto3
1
$>
$> ./do-op toto3 + 4
4
$> ./do-op foo plus bar
0
$> ./do-op 25 / 0
Stop : division by zero
$> ./do-op 25 % 0
Stop : modulo by zero
```

- All the files relative to your program will be into :
  `Piscine_C_J10/do-op/`

`Be careful to the division by 0`

Exercise 3 - my_sort_wordtab

- Write the `my_sort_wordtab` function which sorts by `ascii` order the words you got via `my_str_to_wordtab`
- The sort will be executed by switching the array's pointers.
- It must be prototyped like this:

```
1   int my_sort_wordtab(char **tab);
```

- The function will always return `0`
- Turn-in directory:
`Piscine_C_J10/ex_03/my_sort_wordtab.c`

Exercise 4 - my_advanced_sort_wordtab

- Write the `my_advanced_sort_wordtab` function which will sort the words, depending on the function return value passed by parameter, that you got via `my_str_to_wordtab`
- The sort will be executed by switching the array's pointers.
- It must be prototyped like this:

```
1   int my_advanced_sort_wordtab(char **tab, int(*cmp)(char *, char *));
```

- The function will always return `0`
- Turn-in directory:
`Piscine_C_J10/ex_04/my_advanced_sort_wordtab.c`

Hints `A call to my_advanced_sort_wordtab() with my_strcmp as second parameter will give the same result than my_sort_wordtab()`

Exercise 5 - my_advanced_do-op

- Write a program working almost exactly like the do-op:
  You must include the file `my_opp.h` which will define which function pointer matches with which character.
  This file is located on the intranet with the subject
- At least, you must create 5 functions:
  my_add, my_sub, my_mul, my_div, my_mod, my_usage.
- my_usage displays the possible characters(defined into my_opp.h) like in the example below:

```
1   $> make clean
2   $> make
3   $> ./my_advanced_do-op
4   $> ./my_advanced_do-op 1 + 1
5   2
6   $> ./my_advanced_do-op 1 p 1
7   error : only [ + - / * % ] are supported
8   $> ./my_advanced_do-op 1 +toto 1
9   2
10  $> ./my_advanced_do-op 1 + toto3
11  1
12  $> ./my_advanced_do-op 25 / 0
13  Stop : division by zero
14  $> ./my_advanced_do-op 25 % 0
15  Stop : modulo by zero
```

- You must define the `t_opp` type (corresponding to the `s_opp` structure) allowing your program to compile.
- Don't write into the file `my_opp.h`,even the definition of t_opp. Include your own files if it's necessary.
- Display an error for the operators which don't have correspondence into `my_opp.h`
- Remember that we will probably change the file `my_opp.h` :)
- Turn-in directory:
  `Piscine_C_J10/ex_05/`

Hints `An operator may be composed of several characters`