B1- C-Pool

B-CPE-042

Day 10

Makefile and do-ops

v1.5

Day 10

Makefile and do-ops

repository name: : CPool_Day10
repository rights: : ramassage-tek
language: : C
group size: : 1
allowed functions: : write, malloc, free

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- 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_Day10/lib/my**. For those of you using **.h** files, they must be located in **CPool_Day10/include**. The Autograder will compile your library and your includes.

Create your repository at the beginning of the day and submit your work on a regular basis!
The delivery directory is specified within the instructions for each task.
In order to keep your repository clean, pay attention to `gitignore`.

Task 0

Unit Tests

It is highly recommended to test your functions as you develop them. It is common practice to create a function named `main` (and a designated file to host it) to check the functions separately.
Create a directory named `tests`.
Create a `main` function within a file named `tests-$FUNCTION_NAME.c`, to be stored in the `tests` directory named.
This function must contain all the necessary calls to the task function in order to cover all of the function's possible situations (normal or irregular).

The tests are not applicable for binaries

Task 1

Makefile

Write a **Makefile** that compiles your **libmy**. It should perform the following actions:

- copy the library into **CPool_Day10/lib**,
- copy the my.h into **CPool_Day10/include**,
- implement the **clean** rule.

Your Makefile and all other necessary files (**.c** and **.h**) must be located in **CPool_Day10/lib/my**.

Task 2

do-op

Write a program called *do-op* that computes an operation.
The program must take three arguments:

Terminal

```
~/B-CPE-042> ./do-op value1 operator value2
```

The character operator should correspond to the appropriate function within an array of function pointers (advised but not mandatory, using **if** can be enough).
This directory must have a **Makefile** that includes the following rules: **all, clean, fclean, re**. It must not relink when not necessary.
If the expression is incorrect, the program must display **0**.
If the number of arguments is incorrect, **do-op** must not display anything.

Here are some examples:

Terminal

```
~/B-CPE-042> make clean
~/B-CPE-042> make
~/B-CPE-042> ./do-op
~/B-CPE-042> ./do-op 1 + 1
2
~/B-CPE-042> ./do-op 42friends - ---20toto12
62
~/B-CPE-042> ./do-op 1 p 1
0
~/B-CPE-042> ./do-op 1 +toto 1
2
~/B-CPE-042> ./do-op 1 + toto3
1
~/B-CPE-042> ./do-op toto3 + 4
4
~/B-CPE-042> ./do-op foo plus bar
0
~/B-CPE-042> ./do-op 25 / 0
Stop :  division by zero
~/B-CPE-042> ./do-op 25 % 0
Stop :  modulo by zero
```

All files related to your program must be delivered to: **CPool_Day10/do-op/**

Your main function must return 0.

Task 3

my_sort_wordtab

Write a function that, using **ascii order**, sorts the words received via **my_str_to_wordtab**. The sort should be executed by switching the array's pointers.
The function must be prototyped as follows:

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

The function will always return 0.
Delivery: CPool_Day10/task03/my_sort_wordtab.c

Task 4

my_advanced_sort_wordtab

Write a function that sorts the words depending on the return value of the function passed as parameter, received via **my_str_to_wordtab**. The sort should be executed by switching the array's pointers.
The function must be prototyped as follows:

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

The function will always return 0.
Delivery: CPool_Day10/task04/my_advanced_sort_wordtab.c

A call to **my_advanced_sort_wordtab()** with **my_strcmp** as second parameter, will produce the same result as **my_sort_wordtab()**.

Task 5

my_advanced_do-op

Write a program that functions almost exactly like do-op, except you must include the **my_opp.h** file, which definse which function pointer matches with which character.
The file can be found with the project description on the intranet.
You must create the following functions: **my_add**, **my_sub**, **my_mul**, **my_div**, **my_mod**, **my_usage** (this last one displays the possible characters defined in *my_opp.h*).
Here are some examples:

Terminal

```
~/B-CPE-042> make clean
~/B-CPE-042> make
~/B-CPE-042> ./my_advanced_do-op
~/B-CPE-042> ./my_advanced_do-op 1 + 1
2
~/B-CPE-042> ./my_advanced_do-op 1 p 1
error :  only [ + - / * % ] are supported
~/B-CPE-042> ./my_advanced_do-op 1 +toto 1
2
~/B-CPE-042> ./my_advanced_do-op 1 + toto3
1
~/B-CPE-042> ./my_advanced_do-op toto3 + 4
4
~/B-CPE-042> ./my_advanced_do-op 25 / 0
Stop :  division by zero
~/B-CPE-042> ./my_advanced_do-op 25 % 0
Stop :  modulo by zero
```

You must define the **t_opp** type, which corresponds to the **s_opp** structure, in order for your program to compile.
Do not write, even the definition of **t_opp**, in the **my_opp.h** file.
Include your own file if necessary.
Display an error for the operators that do not correspond to **my_opp.h**.

Delivery: CPool_Day10/task05/

Remember we will probably change the **my_opp.h** file.... and an operator may be composed of several characters.

Your main function must return `0`.