B1 - Unix & C Lab Seminar

B-CPE-100

Day 10

Makefile and do-ops

1.0

Day 10

language: C

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

- Don't push your `main` function into your delivery directory, we will be adding our own. Your files will be compiled adding our `main.c`.
- 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 must be found in `lib/my/`. For those of you using .h files, they must be located in `include/` (like the `my.h` file).

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`.

Allowed system function(s): `write`, `malloc`, `free`

We still encourage you to write unit tests for all your functions!
Check out Day06 if you need an example, and re-read the guide.

TASK 01 - MAKEFILE

Delivery: lib/my/*

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

- copy the built library into `lib/`,
- copy the `my.h` into `include/`,
- implement the `clean` rule.

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

Don't re-use the `build.sh` script previously made. It will be deleted before using your Makefile.
Everything must be done directly by the Makefile.

TASK 02 - DO-OP

Delivery: do_op/*

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

```
Terminal
~/B-CPE-100> ./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` and exit as an error.
If the number of arguments is incorrect, `do-op` must not display anything and exit as an error.
Here are some examples:

```
Terminal
~/B-CPE-100> make clean
~/B-CPE-100> ./do-op
~/B-CPE-100> ./do-op 1 + 1
2
~/B-CPE-100> ./dop-op 42friends - ---20toto12
62
~/B-CPE-100> ./do-op 1 p 1
0
~/B-CPE-100> ./do-op 1 +toto 1
2
~/B-CPE-100> ./do-op 1 + toto3
1
~/B-CPE-100> ./do-op toto3 + 4
4
~/B-CPE-100> ./do-op foo plus bar
0
~/B-CPE-100> ./do-op 25 / 0 > /dev/null
Stop: division by zero
~/B-CPE-100> ./do-op 25 % 0 > /dev/null
Stop: modulo by zero
```

Your main function must return `0` if everything is alright and `84` in case of errors.

TASK 03 - MY_SORT_WORD_ARRAY

Delivery: my_sort_word_array.c

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

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

The function will always return 0.

TASK 04 - MY_ADVANCED_SORT_WORD_ARRAY

Delivery: my_advanced_sort_word_array.c

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

```
int my_advanced_sort_word_array(char **tab, int(*cmp)(char const *, char const *));
```

The function will always return 0.

A call to `my_advanced_sort_word_array` with `my_strcmp` as second parameter, will produce the same result as `my_sort_word_array`.

TASK 05 - MY_ADVANCED_DO-OP

Delivery: my_advanced_do_op/*

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

Terminal

```
~/B-CPE-100> make clean
~/B-CPE-100> make
~/B-CPE-100> ./my_advanced_do-op
~/B-CPE-100> ./my_advanced_do-op 1 + 1
2
~/B-CPE-100> ./my_advanced_do-op 1 p 1 > /dev/null
error: only [ + - / * % ] are supported
~/B-CPE-100> ./my_advanced_do-op 1 +toto 1
2
~/B-CPE-100> ./my_advanced_do-op 1 + toto3
1
~/B-CPE-100> ./my_advanced_do-op toto3 + 4
4
~/B-CPE-100> ./my_advanced_do-op 25 / 0 > /dev/null
Stop: division by zero
~/B-CPE-100> ./my_advanced_do-op 25 % 0 > /dev/null
Stop: modulo by zero
```

You must define the `struct operator` type in a header file of your choice that you will put in the `include/` folder and be included in your program accordingly in order for your program to compile.
Display an error for the operators that do not correspond to one defined in `my_opp.h`.

We will use our own modified version of the `my_opp.h` file.... and an operator may be composed of several characters.

Your main function must return `0` if everything is alright and `84` in case of errors.