B1- C Pool

B-CPE-042

Day 08

Compilation, Allocation

v1.6

Day 08

Compilation, Allocation

repository name: : CPool_Day08
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,...).
- 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 is found in **CPool_Day08/lib/my**. For those of you using **.h** files, they must be located in **CPool_Day08/include**.

The Autograder will compile your functions the following way:

Terminal

```
~/B-CPE-042> cd task01
~/B-CPE-042> cc *.c -c -I../include/
~/B-CPE-042> cc *.o ~autograder/main_task01.o -L../lib/my/ -o task01 -lmy
```

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).

Always check the empty strings and *int*'s special values (0, MIN, MAX)!

Task 1

my_strdup

Write a function that allocates memory and copies the string given as argument in it.
It must be prototyped as follows:

```
char *my_strdup(char *src);
```

The function must return a pointer to the newly allocated string.
Delivery: CPool_Day08/task01/my_strdup.c

Task 2

sum_params

Write a function that turns the command-line given arguments into a single string. Arguments are to be separated by '\n'. The function will be called the following way:

```
int main(int ac, char **av)
{
        my_putstr(sum_params(ac, av));
        return (EXIT_SUCCESS);
}
```

Think about using a **recursive grep** in /usr/include... Maybe the compiler would be happier with that? ;-)

Terminal

```
~/B-CPE-042> cc -o sum_params sum_params.c main.c my_putstr.c
~/B-CPE-042> ./sum_params toto titi | cat -e
./sum_params$
toto$
titi
```

The function must be prototyped as follows:

```
char *sum_params(int argc, char **argv);
```

Delivery: CPool_Day08/task02/sum_params.c

Task 3

my_show_wordtab

Write function that displays the content of an array of words.
There must be one word per line, and each word must end with '\n', including the last one.
The function must be prototyped as follows:

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

Delivery: CPool_Day08/task03/my_show_wordtab.c

Here is an example of main function:

```
int main()
{
        char *my_array [] = {"The", "Answer", "to", "the", "Great", "Question...", "Of", "Life,",
                "the", "Universe", "and", "Everything...", "Is...", "Forty-two,", 0};

        my_show_wordtab(my_array);
}
```

Task 4

my_str_to_wordtab

Write a function that splits a string into words. Separators will all be non-alphanumeric characters.
The function returns an array in which each cell contains the address of a string (representing a word).
The last cell must be null to terminate the array.
The function must be prototyped as follows:

```
char **my_str_to_wordtab(char *str);
```

Delivery: CPool_Day08/task04/my_str_to_wordtab.c

Task 5

convert_base

Write a function that returns the result from the **nbr** string conversion (expressed in a **base_from** radix to a **base_to** radix), in the form of a newly, and sufficiently, allocated string.
The number, represented by **nbr**, fits in an integer.
The function must be prototyped as follows:

```
char *convert_base(char *nbr, char *base_from, char *base_to);
```

Delivery: CPool_Day08/task05/convert_base.c