C - Pool - Tek1

Day 08 subject

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.
- We collect all the .c files from your turn in folder and will compile with your `libmy` found in `Piscine_C_J08/lib/my`.
- For those who use .h files, they must be found in:
`Piscine_C_J08/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
[...]
```

- All the functions needed to compile the exercise must be found either in the `.c` file specified by the subject or in your `lib`.
- The function asked in the subject must be in the required `.c` file.
- For example, if you use `my_strdup` for `my_str_to_wordtab`, `my_strdup` shall be found in the `.c` file of `my_str_to_wordtab` because it cannot be in your library since required in exercise 1.
- Turn in folder:
`Piscine_C_J08`

 *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

```
Here is a partial list of tests:
- Check the empty strings
- Check the pointer's values
```

1 - my_strdup

- Write a function that allocates enough memory and makes a copy of the string given as argument.
- It shall be prototyped as follows:

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

- It must return a pointer on the newly-allocated string.
- Turn in folder:
  `Piscine_C_J08/ex_01/my_strdup.c`

Hints `Hint: man strdup`

2 - convert_base

- Write a function that returns the result of the conversion from the string `nbr` 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.
- It shall be prototyped as follows:

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

- Turn in folder:
`Piscine_C_J08/ex_02/convert_base.c`

3 - sum_params

- Write a function that turns the command-line given arguments into a single string. Arguments shall be '\n'-separated.
- Example:

```
1  int main(int ac, char **av)
2  {
3      my_putstr(sum_params(ac, av));
4      return (EXIT_SUCCESS);
5  }
```

Indices What about a recursive grep in /usr/include, maybe the compiler would be happier ;)

```
1  (foo_b@fedora) cc -o sum_params sum_params.c main.c my_putstr.c
2  (foo_b@fedora) ./sum_params toto titi | cat -e
3  ./sum_params$
4  toto$
5  titi(foo_b@fedora)
```

- It shall be prototyped as follows:

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

- Turn in folder:
`Piscine_C_J08/ex_03/sum_params.c`

4 - my_str_to_wordtab

- Write a function that splits a string into words.
- Separators are all non-alphanumeric characters.
- The function returns an array, where each cell contains the adress of a string representing a word. The last element shall equal 0, thus marking the end of the array.
- The transmitted string will be modifiable and shall be modified in your function.
- It shall be prototyped as follows:

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

- Turn in folder:
`Piscine_C_J08/ex_04/my_str_to_wordtab.c`

5 - my_show_wordtab

- Write a function that displays the content of the array created in the previous exercise.
- There shall be one word per line.
- Each word shall be '\n'-terminated, including the last one.
- This exercise will be compiled with your my_str_to_wordtab.c
- Be careful not to have `multiple define`.
- It shall be prototyped as follows:

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

- Turn in folder:
`Piscine_C_J08/ex_05/my_show_wordtab.c`