B1 - Phoenix Bootcamp

B-BOO-101

Day 03

Phoenix library and arguments

1.1

Day 03

repository name: BOO_phoenix_d03_*$ACADEMICYEAR*
repository rights: ramassage-tek
language: C

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- 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).

Unless stated otherwise, don't push your `main` functions into your delivery directory. Your files will be compiled adding our own `main`.

All `.c` files from your delivery folder will be collected and compiled with your `libphoenix`, which is to be found in `/lib/phoenix`. Header files (`.h`) must be located in `includes/`.

Allowed system function(s): write.

TASK 01 - LIBPHOENIX.A

Delivery: `lib/phoenix/Makefile`, `lib/phoenix/ALL_THE_NEEDED_SOURCE_FILES.c`

Create a `Makefile` that, when called, build a static library called `libphoenix.a` in your `lib/phoenix` directory. This library must contains, at least, the following functions:

```
void my_putchar(char c);
int show_number(int nb);
int show_string(char const *str);
char *reverse_string(char *str);
int to_number(char const *str);
int is_prime_number(int nb);
char *my_strcpy(char *dest, char const *src);
int my_strncmp(char const *s1, char const *s2, int n);
char *my_strstr(char *str, char const *to_find);
```

Beware to build your library in the correct folder because it will be used to build all of the following tasks.

TASK 02 - PHOENIX.H

Delivery: `includes/phoenix.h`

Write your `phoenix.h` header file that contains the prototypes of all the functions exposed by your `libphoenix.a`.

To check exposed functions, see the man of `nm`.

Beware to place your header in the correct folder because it will be used to build all of the following tasks.

TASK 03 - CONCAT_STRINGS

Delivery: `concat_strings.c`

Write a function that concatenates two strings. It must be prototyped the following way:

```
char *concat_strings(char *dest, char const *src);
```

`man strcat`

Your function will be built using your own library.

Terminal

```
~/B-BOO-101> make -C lib/phoenix
~/B-BOO-101> gcc *.c -Iincludes -Llib/phoenix -lphoenix
```

TASK 04 - SHOW_PARAMS

Delivery: `task04/show_params.c`, `task04/Makefile`

Write a **program** called `show_params` that displays its arguments (received on the command line). Since it is a **program**, you need to put the `main` function in your delivered file.
You are to display all arguments (including `argv[0]`, on different lines.)
Your program will be built by calling your `Makefile`.

Your main function must return `0`.

```
Terminal
~/B-BOO-101> cd task04
~/B-BOO-101> make clean && make fclean && make && make re
~/B-BOO-101> ./show_params test "This is a test" retest | cat -e
./show_params$
test$
This is a test $
retest$
~/B-BOO-101> echo $?
0
```

TASK 05 - SORTED_PARAMS

Delivery: `task05/sorted_params.c`, `task05/Makefile`

Write a **program** called `sorted_params` that displays all its arguments, in `ASCII` order.
You are to display all arguments (including `argv[0]`, on different lines.)
Your program will be built by calling your `Makefile`.

Your main function must return `0`.

```
Terminal
~/B-BOO-101> cd task05
~/B-BOO-101> make clean && make fclean && make && make re
./sorted_params test "This is a test" retest | cat -e
./sorted_params$
This is a test $
retest$
test$
```