B1 - Phoenix Bootcamp

B-BOO-101

Day 04

Memory, re-parameters and structures

1.1

Day 04

repository name:	BOO_phoenix_d04_*$ACADEMICYEAR*
repository rights:	ramassage-tek
language:	C
compilation:	make -C lib/phoenix && gcc *.c -lincludes -Llib/phoenix -lphoenix (+ our own files containing the `main` function)

- 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`, `malloc`, `free`.

TASK 01 - DUPLICATE_STRING

Delivery: `duplicate_string.c`

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

```
char *duplicate_string(char const *src);
```

The function must return a pointer to the newly allocated string.

TASK 02 - CONCAT_PARAMETERS

Delivery: `task02/concat_parameters.c`, `task02/Makefile`

Write a **program** called `concat_parameters` that turns the command-line given arguments into a single string. Arguments are to be separated by `\n`. Do not submit a `main` function in your delivery, we will be adding our own `main` in the file `task02/main.c` before calling your `Makefile`.
Our `main.c` file is as follows:

```
#include <stdlib.h>
#include "phoenix.h"

int main(int ac, char **av)
{
    char *str = concat_parameters(ac, av);

    show_string(str);
    free(str);
    return (EXIT_SUCCESS);
}
```

Terminal

```
~/B-BOO-101> cp /path/to/somewhere/main.c task02/main.c && cd task02
~/B-BOO-101> make clean && make fclean && make && make re
~/B-BOO-101> ./concat_parameters toto titi | cat -e
./concat_parameters$
toto$
titi
```

TASK 03 - SPLIT_STRING

Delivery: `split_string.c`

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 **split_string(char const *str);
```

TASK 04 - SHOW_STRING_ARRAY

Delivery: `show_string_array.c`

Write a 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 show_string_array(char * const *array);
```

Here is an example of main function:

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

  show_string_array(test_word_array);
}
```

TASK 05 - PARAMETERS_TO_ARRAY

Delivery: `parameters_to_array.c`

Write a function that stores the program's parameters into an array of structures and returns the address of the array's first cell. All array elements are to be addressed, including `av[0]`.
The function must be prototyped as follows:

```
struct info_param *parameters_to_array(int ac, char **av);
```

The structures contained is the array are to be allocated.
To indicate the end of the array, the `str` field of its last cell must be set to `0` or `NULL`.
The structure is defined as follows:

```
struct info_param
{
  int length;          // parameter's length
  char *str;           // parameter's address
  char *copy;          // parameter's copy
  char **word_array;   // the result of split_string(str)
};
```

Do not submit the `struct info_param` structure; the tests set will use its own.

Your function will be tested with your own `split_string`.

TASK 06 - SHOW_PARAMETERS_ARRAY

Delivery: `show_parameters_array.c`

Write a function that displays the content of an array created with the previous function, and prototyped as follows:

```
int show_parameters_array(struct info_param const *par);
```

For each cell, display the following elements (one per line): parameter, size and words (one per line).

Do not submit the `struct info_param` structure; the tests set will use its own.