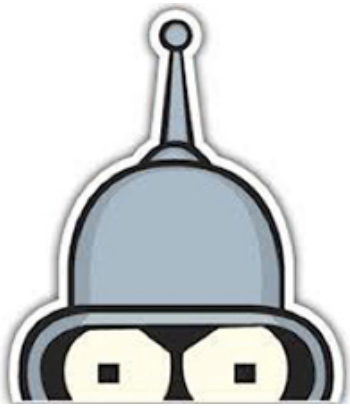
B1 - Elementary Programming in C

B-CPE-111

Get Next Line

a very first parsing tool



2.0

{EPITECH.}

Get Next Line

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

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.

The only authorized functions are: **read, malloc and free**.

The goal of this project is to write a function that returns a read line from a file descriptor.
If there are no more lines to return, or if there is an error during the reading, the function will come back **NULL**.

You must define a macro called *READ_SIZE* in your **get_next_line.h** file, which indicates the amount of characters to be read for each **read()** call.

```
#   define READ_SIZE (/* value here */)
```

The value of this macro may be changed (by another strictly positive value) during the evaluation in order to verify that you are using it correctly.

Don't forget the **#ifndef** directive to avoir re-defining the macro.

A static variable may be used to save some datas between each call to **get_next_line**.

The function should be prototyped as follows:

```
char *get_next_line(int fd);
```

get_next_line must return the results **without** the '\n'.

You must submit **get_next_line.c** and **get_next_line.h** (both of which must be located at the directory's root), but no **main** function.

A Makefile is useful only if you do unit tests. That is with the *tests_run* rule.

EXAMPLE

Here is an example of use of the function:

```
#include "my.h"
#include "get_next_line.h"

int main(void)
{
    char *s = get_next_line(0);

    while (s) {
        my_putstr(s);
        my_putchar('\n');
        free(s);
        s = get_next_line(0);
    }
    return (0);
}
```

TESTS EXAMPLES

The following example do only one test by calling the get_next_line function. Be sure to already test all internal functions you may have.

```c
#include <criterion/criterion.h>
#include <criterion/redirect.h>
#include <fcntl.h>
#include <unistd.h>

char *get_next_line(int fd);

int fd = -1;

/*
The file contain the following content:
Confidence is so overrated.
It's when we're most uncomfortable and in desperate need of an answer that we grow
    the most.
*/

void open_file(void)
{
    fd = open("tests/data.txt", O_RDONLY);
    cr_redirect_stdout();
}

void close_file(void)
{
    if (fd != -1)
        close(fd);
}

Test(get_next_line, read_line, .init = open_file, .fini = close_file) {
    char *expected = "Confidence is so overrated.";
    char *got = get_next_line(fd);
    cr_assert_str_eq(got, expected);
}
```

The get_next_line function can be particularly hard to unit test correctly. have you identified why? It's a very good subject to talk about during follow-ups and reviews!

Speaking of reviews, you could also present a version of get_next_line who doesn't have the same flaws, as a bonus.