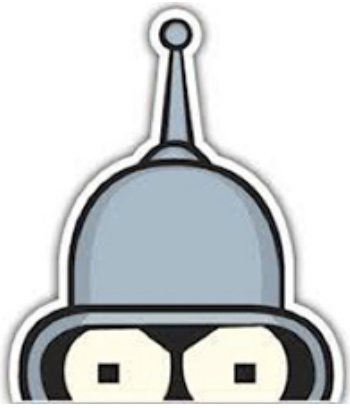
B1 - Elementary Programming in C

B-CPE-111

Bootstrap

Get Next Line



3.0

Bootstrap

language: C

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.

Each exercise is to be submitted in a C file with the same name as the function in the repository's root. Each file will be separately compiled with our main function.

The challenge of the GNL project is being able to read a file, line by line, without knowing the file size or the size of the lines.

We will begin with a little reminder of how to read data in a given file; we will then work on a new C language tool: static variables.

If you aren't yet comfortable with the file descriptor concept, take a few minutes to go over it again.

In the pages after all the steps, you will find snippets of unit tests that can help you write more unit tests to test your functions.

Reading Characters

Step 1

Write a function that both reads and returns the next *n* available characters, *n* being passed as parameter:

```
char *read_next_n_bytes(int fd, int n);
```

The *n* returned characters must be stored in a new memory zone that can be free'd.
In the event of an error, or if there is nothing left to read, return **NULL**.

Step 2

Write a function that takes a file descriptor as parameter and makes as many calls to the function **read** (reading *n* characters each time) as necessary in order to read a line and display it on the standard output. The '\n' should not be displayed.

```
void read_and_display_read_line_n(int fd, int n);
```

The extra read characters will not be displayed.

Step 3

We will go back over Step 3 and add an extra obstacle: return the read line (into a newly allocated memory zone) and display the extra read characters on the standard output.

```
char* read_line_and_display_remaining(int fd, int n);
```

The returned address must be passed as parameter to **free**.

Static variables

Step 1

Write a function that returns the number of times it was called since the beginning of the program.

```
int get_nb_calls(void);
```

Step 2

Write a function that returns the number of times it was called since the beginning of the program. However, this time, program your static variable so that the intial value is 9.
The first system call should then increase to 10, the following to 11,…

```
int get_nb_calls_init_9(void);
```

Step 3

For the last step of this Bootstrap, write a function that returns the string's content, *"Hello my name is Chucky. Do you want to be my friend?"*, in chunks of 5 characters.
The first system call should return "Hello", the next " my n",…

In order to do this exercise correctly, don't create an absurd forest of if/else; instead, use a static variable.

```
char *get_sentence_chunk_by_chunk(void);
```

You must return the chunks of the sentence to a new memory zone each time (which can be free'd).
If you have come to the end of the sentence, you must return **NULL**.

UNIT TESTS EXAMPLES

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

int get_nb_calls(void);
char *read_line_and_display_remaining(int fd, int x);

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(read_line_and_display_remaining, line_read, .init = open_file, .fini =
    close_file) {
    char *expected = "Confidence is so overrated.";
    char *got = read_line_and_display_remaining(fd, 6);
    cr_assert_str_eq(got, expected);
}

Test(read_line_and_display_remaining, display_remaining, .init = open_file, .fini =
    close_file) {
    read_line_and_display_remaining(fd, 6);
    cr_assert_stdout_eq_str("It");
}

Test(get_nb_calls, one_call) {
    int ret = get_nb_calls();
    cr_assert_eq(ret, 1);
}

Test(get_nb_calls, two_calls) {
    int ret = get_nb_calls();
    cr_assert_eq(ret, 1);
    ret = get_nb_calls();
    cr_assert_eq(ret, 2);
}
```