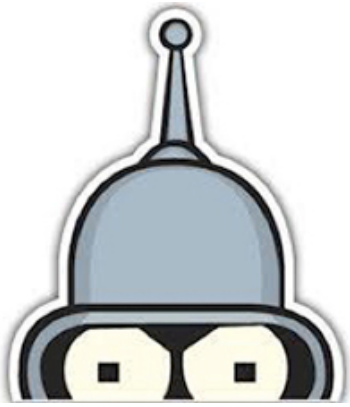
B1 - Elementary Programming in C

B-CPE-110

Bootstrap

Matchstick



3.0

{EPITECH.}

Bootstrap

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).

Each exercise is to be submitted in a file called "**name_of_the_function.c**" at the root of the repository. Each file will be compiled separately with our main function.

If you haven't already done so, please read the Matchstick project description.
If you didn't come to the Matchstick Kick-off, well, you should have!

Matchstick is a basic game. To invent more complex and successful games, you need to familiarize yourself with the basic elements of a game:

- the game board,
- rules,
- player(s)
- air-tight error management (no one likes glitches in games).

Your project will, therefore, consist of a game loop that will enable players to progress, or make the game board progress, by following the rules in order to win (or lose…).
In this bootstrap, we will focus on handling the game board and interactions with the player.

STEP 1

Write a function that displays a game board with 4 lines of matchsticks, in pyramid form. The matches will be represented by the character '|', with the following prototype:

```
void print_game_board(void);
```

It should output this:

```
*********
*   |   *
*  |||  *
* ||||| *
*|||||||*
*********
```

Make sure to not have any empty, useless spaces. In other words, the last matchstick line shouldn't have any spaces between the matches and the frame.

STEP 2

When a player (whether AI or human) removes a certain number of matches, the game board updates what is displayed. Write a function which, starting from a 4 line game board (like the one you generated in the previous step), takes as parameter the line and the number of matches to be removed and displays the updated game board, with the following prototype:

```
void print_updated_board_game(int line, int nb_matches);
```

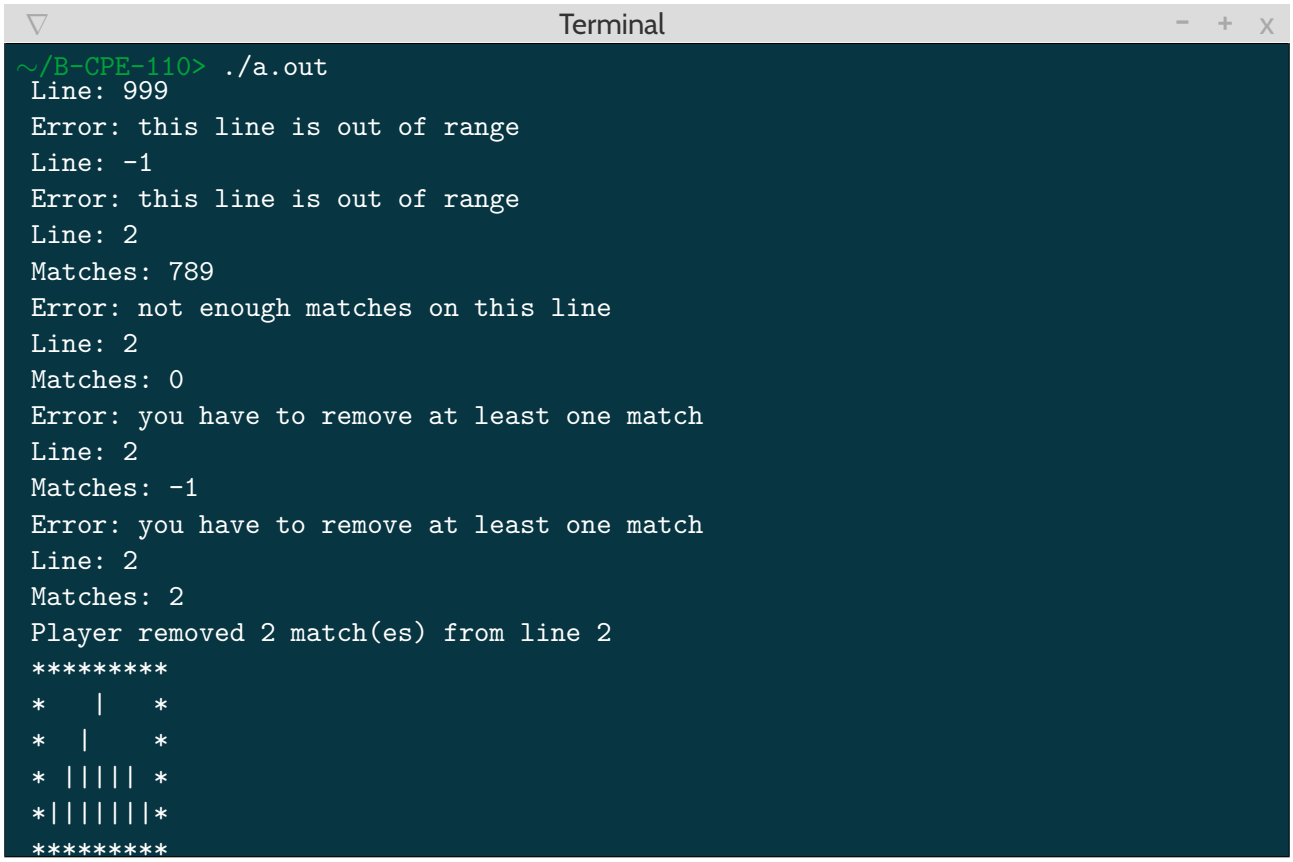
STEP 3

Now let's create real player interaction. Write a function that read the number of lines and the number of matches on the standard input and displays the updated game board. Make sure to implement error management, as was indicated in the project overiew. The function should be prototyped as follows:

```
void read_player_move_and_print_updated_board_game(void);
```

Are your get_line implementation and my_getnbr function ready and perfectly functional?

For example:


Terminal
~/B-CPE-110> ./a.out
Line: 999
Error: this line is out of range
Line: -1
Error: this line is out of range
Line: 2
Matches: 789
Error: not enough matches on this line
Line: 2
Matches: 0
Error: you have to remove at least one match
Line: 2
Matches: -1
Error: you have to remove at least one match
Line: 2
Matches: 2
Player removed 2 match(es) from line 2
****
* | *
* | *
* ||||| *
|||||||
****