Week 8

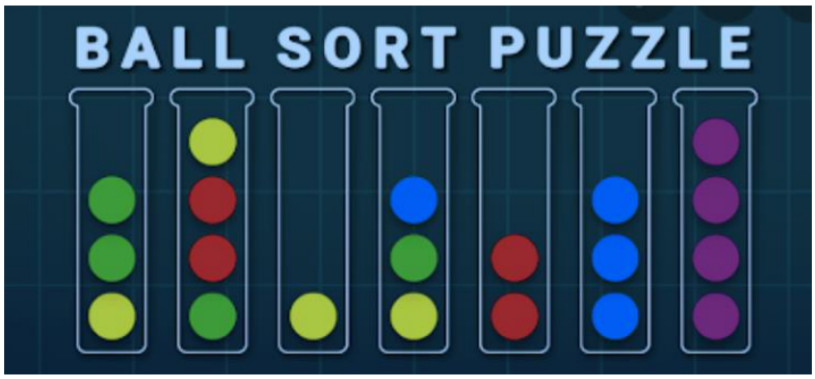
Problem Set 4: Ball Sort Puzzle

Goals

1. Create custom features as specified.
2. Work routinely with two-dimensional arrays and strings.
3. Use fields as parameters (input and output) in functions.
4. Use a random number generator.

Ball Sort Puzzle

Probably every one of you has already played or heard of a game called Ball sort puzzle. The game starts with balls of different colors swept in several reels. The aim of the game is for the player to gradually place them in reels according to color. However, the player may not place balls of colors other than their own. If all the balls are stacked, the player wins.



The game proceeds as follows:

1. The player gets a field where the characters are scattered.
2. Gradually moves characters to places where they can be placed.
3. If they are all arranged, the player wins.

Colored balls would definitely be nice in our solution, but a terminal interface will suffice. We will use characters instead of colors and balls.

Your task will be to program five functions:

- `void generator(const int rows, const int columns, char field[rows][columns])`- Generating a two-dimensional game field.
- `void down_possible(const int rows, const int columns, char field[rows][columns], int x, int y)`- A character can go to the same character and get to the bottom of the column.
- `bool check(const int rows, const int columns, char field[rows][columns])`- Check the same columns.
- `void game_field(const int rows, const int columns, char field[rows][columns])`- Game field rendering.
- `ball_sort_puzzle()`- Functionality of the game itself.

All 5 functions will be located in the file `ballsortpuzzle.c`. Their declarations are listed in the file `ballsortpuzzle.h`. The main program is located in the file `main.c`.

In addition to the required functions, you can also create your own functions. However, these will be private to your module (they will not be declared in the header file `ballsortpuzzle.h`).

Task 1: Generate a field

Program the function `void generator(const int rows, const int columns, char field[rows][columns])`with parameters:

- `const int rows`- The number of rows in the field
- `const int columns`- Number of columns in the field
- `char field[rows][columns]`- A two-dimensional field in which characters will be stored

The function **does return** not any value.

The function randomly generates a two-dimensional array of characters. However, they must comply with the following rules:

- There must always be 2 empty columns.
- Each character can occur just as many times as the height of the column. It is enough for us that this number is `rows`.
- Characters must be generated randomly, no columns already filled in.
- The player (foreground) indexes from number 1 and from his point of view the playing field starts at the top left. The game (background) indexes from the number 0 and from its point of view the playing field also starts at the top left.

Example 1:

```
        FOREGROUND                          BACKGROUND              SOME INDEXES
1 | @ | * |   |   | + | + |    [0]: {'@', '*', ' ', ' ', '+', '+'}    [0][4]: '+'
2 | * | + |   |   | * | + |    [1]: {'*', '+', ' ', ' ', '*', '+'}    [1][2]: ' '
3 | @ | ^ |   |   | ^ | @ |    [2]: {'@', '^', ' ', ' ', '^', '@'}    [2][4]: '^'
4 | ^ | ^ |   |   | @ | * |    [3]: {'^', '^', ' ', ' ', '@', '*'}    [3][5]: '*'
   --- --- --- --- --- ---
    1   2   3   4   5   6            [0]  [1]  [2]  [3]  [4]  [5]
```

Task 2: Move characters around the playing field

Program the function `void down_possible(const int rows, const int columns, char field[rows][columns], int x, int y)` s paramerami:

- `const int rows`- The number of rows in the field
- `const int columns`- Number of columns in the field
- `char field[rows][columns]`- The field in which the characters are stored
- `int x`- The column number from which we want to move the character
- `int y`- The column number where you want to move the character

The function **does return** not any value.

This function will be used to move characters around the board. In addition to the transfer, he must ensure that the character cannot get to a character other than himself.

Attention should also be paid to the situation in which the character will not be moved to the character, but to the bottom of the column.

If a player tries to move a character to an inappropriate location, the function does nothing and notifies the player that he is trying to make an unauthorized move.

When inserting a character into a column, make sure that the character is placed at the bottom of the bottom so that it does not float in the air.

The player may not select the same column to move, for example from column 2 to column 2.

Example of moving from column 1 to column 2 (bottom of vessel):

```
1 | * |   | ^ | @ | @ |   |    1 |   |   | ^ | @ | @ |   |
2 | + |   | ^ | @ | ^ |   |    2 | + |   | ^ | @ | ^ |   |
3 | + |   | * | + | + |   |    3 | + |   | * | + | + |   |
4 | * |   | * | @ | ^ |   |    4 | * | * | * | @ | ^ |   |
   --- --- --- --- --- ---        --- --- --- --- --- ---
    1   2   3   4   5   6          1   2   3   4   5   6
```

Task 3: Checking the arrangement of columns

Program the function `bool check(const int rows, const int columns, char field[rows][columns])`with parameters:

- `const int rows`- The number of rows in the field
- `const int columns`- Number of columns in the field
- `char field[rows][columns]`- The field in which the characters are stored

The function **returns a** value `true`if all characters in the columns are the same. The function **returns a** value `false`if not all characters are the same. Be aware of the situation if there is only one character in the column. This situation is not considered the final state of the game. Each column must have all the same characters up to the top.

The function returns `false`:

```
1 |   |   | ^ |   |   |   |
2 | + |   | ^ | @ | ^ |   |
3 | + |   | * | + | + | @ |
4 | * | * | * | @ | ^ | @ |
   --- --- --- --- --- ---
    1   2   3   4   5   6
```

The function returns `true`:

```
1 |   | @ | + | * |   | ^ |
2 |   | @ | + | * |   | ^ |
3 |   | @ | + | * |   | ^ |
4 |   | @ | + | * |   | ^ |
   --- --- --- --- --- ---
    1   2   3   4   5   6
```

Task 4: Game field

Program the function `void game_field(const int rows, const int columns, char field[rows][columns])`with parameters:

- `const int rows`- The number of rows in the field
- `const int columns`- Number of columns in the field
- `char field[rows][columns]`- The field in which the characters are stored

The function **does return** not any value.

This feature is left to your imagination. It will be used to draw the game field. The more creative, the better. For enthusiasts of the curses library, this library will be used in the translation of the program.

Task 5: Game

Program the function `void ball_sort_puzzle()`which will control the entire operation of the game.

The function **does return** not any value.

You will use the functions you have created in this function `bool check()`, `void down_possible()`, `void generator()` a `void game_field()`.

The playing field must be generated first. If it is generated, you will render it and add a user interface in the form of a dialog.

You first ask the player for the column from which you want to move the character and then for the column where you move it. You move this information into function `void down_possible()`. After each move, the status of the game will be verified by the function `bool check()`. It's up to you how the game ends. It can be terminated, for example, by typing the text: `Congratulations! You won!`

Example of game progress 1:

```
$  ./ballsortpuzzle
 1 | * | @ |   |   |   |   |
 2 | * | @ |   |   |   | ^ |
 3 | + | @ |   | + | * | ^ |
 4 | ^ | @ | + | * | + | ^ |
    --- --- --- --- --- ---
     1   2   3   4   5   6

 Enter what:  4

 Enter where:  2
 MUST BE SAME

 1 | * | @ |   |   |   |   |
 2 | * | @ |   |   |   | ^ |
 3 | + | @ |   | + | * | ^ |
 4 | ^ | @ | + | * | + | ^ |
    --- --- --- --- --- ---
     1   2   3   4   5   6
```

```
Enter what:  4

Enter where:  3

1 | * | @ |   |   |   |   |
2 | * | @ |   |   |   | ^ |
3 | + | @ | + |   | * | ^ |
4 | ^ | @ | + | * | + | ^ |
   --- --- --- --- --- ---
    1   2   3   4   5   6
```