Mandatory tasks

1. Let's make a minesweeper game! The size of the track should be fixed at 10
    - 10, the rows should be marked from A to J, the columns should be from 0 to 9, the fields should be blank at the beginning. 0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J During the program we will need two tables, one in which we store the data and one in which we only store the revealed fields (in a few exercises we can simplify this by using struct;) Write a function that plots a table passed in a parameter as above.
2. The track should be a 10x10 integer matrix. Try it with one- and two-dimensional array representations!
3. Write a function that randomly places N mines on the board, being careful not to place them where there is already a mine. The number of mines is requested as a command parameter and passed to the previous function. The number of mines should not be less than 3 and not more than 30!
4. Write a function that fills the "numbers" on the field, that is, for any field that is not a mine, it fills how many mines there are in adjacent fields.
5. Then ask the player for coordinates until the mines run out or one of them explodes. The requested data should be in A9 format, so we know that the first character is the row and the second is the column coordinate.

Optional tasks

1. Add save and retrieval options as an extra feature, if the player enters the "save <filename>" command, save the current job to the file with the specified name, if you issue the "load <filename>" command, load the game position from that file. Remember that we need to save the game board with all the data as well as the game board revealed by the player! Consider in what form you should save the data: - complete board drawn, - only the position of the mines from which we can reconstruct the board.

Advanced tasks

1. Think about how we can get the player to not be able to “cheat,” i.e., by reading the saved game position, he can’t easily guess where the mines are.