COMP2129 Assignment 1

Due: 11:59pm Friday 25th March

Task

For this assignment, you are required to implement in C the game commonly known as “Minesweeper"[1]. Note that this game is commonly found in Windows and Linux installations; have a look and play the game so that you get an understanding of the task. In each game, mines are hidden in a two-dimensional minefield and the object of the game is to uncover the empty (safe) tiles and flag the tiles containing mines.

In our implementation, a 4×4 minefield will look like this:

```
+----+
|****|
|****|
|****|
|****|
+----+
```

You should read the size of the minefield and the placement of the mines from standard input using **`scanf()`**. After the initial configuration of the minefield is read in, you should print the minefield to standard output before every turn using **`printf()`** and **`putchar()`**, and then read a command from standard input. The player will interact with the game by typing commands such as **`u x y`** to uncover the tile with coordinates (x, y) or **`f x y`** to flag the tile. The game is won when all mines have been flagged and all empty tiles have been uncovered. When this happens, your program should print **`won`**. The game is lost if the player uncovers a tile with a mine, when this happens your program should print **`lost`**.

At the beginning of the game, all tiles are displayed as `*` (see the example above). Tiles that have been flagged are displayed as **`f`**, and tiles that have been safely uncovered display the number between 0 and 8 which represents the number of neighbouring mines.

Your implementation will follow along the lines of the game in Windows and Linux with one exception: we simplify the game such that uncovering one field does not automatically uncover neighboring fields if no mines are in its proximity, i.e., the action of uncovering uncovers a single field rather than a multitude. In addition, a player cannot remove a flag that was previously set.

The number of mines for each game is fixed to 10. The maximal size of the grid is 100×100 fields but it can be smaller. The number of fields in a grid must be greater than 10. Use two-dimensional arrays to describe the game state (do not use data structures in dynamic memory).

[1] http://en.wikipedia.org/wiki/Minesweeper_(video_game)

Input

The input consists of two parts: the first part defines the game and the second part describes the moves of the player. The input is defined as a sequence of instructions, and each instruction is on a new line. Each instruction causes a specific action. We have four instructions:

1. Define the grid size with following command: **`g <WIDTH> <HEIGHT>`**, where **`<WIDTH>`** and **`<HEIGHT>`** are integers between 1 and 100. This command sets the size of the grid to a **`<WIDTH>`** $\times$ **`<HEIGHT>`** grid. It is the first command in the sequence and can only be issued once. If the grid instruction is successful, it will print itself on standard output. Otherwise it will print **`grid-error`**, and terminate.

2. Mine placement: **`b <x> <y>`**. This command places a mine at location (x, y) where the coordinates range from $(0, 0)$ to $(\text{<WIDTH>} - 1, \text{<HEIGHT>} - 1)$. There will always be ten occurences of this instruction, and they will always follow the **`g`** command. If the mine-placement instruction is successful, it will print itself on standard output. Otherwise it will print **`mine-error`**, and terminate.

3. Uncover a tile: **`u <x> <y>`**. This command uncovers a field at location **`(<x>,<y>)`**. If the uncovering instruction is successful, it will print itself on standard output. Otherwise it will print **`uncover-error`**, and terminate.

4. Place a flag: **`f <x> <y>`**. This command sets a flag at location (x, y). If the flag placement instruction is successful, it will print itself on standard output. Otherwise it will print **`uncover-error`**, and terminate.

5. If none of the above instruction show up in the input sequence, or with the wrong number of parameters, exit the program and print **`input-error`**. Hint: use **`scanf("%c %d %d")`** to read the sequence of instructions.

All fields must be either uncovered or a flag must be placed. If the instruction sequence terminates before, the program should exit and print **`missing-input`**. The player can only set 10 flags. After placing the last mine, the grid is printed between every instruction of the player's moves. If the player has successfully uncovered all fields and set the flags, the program prints **`won`** otherwise the program prints **`lost`**. For example if you have the input:

```
g 3 4
b 0 0
b 0 1
b 0 2
b 0 3
b 1 0
b 1 1
b 1 2
b 1 3
b 2 0
b 2 3
u 2 1
u 2 2
f 0 0
f 0 1
f 0 2
f 0 3
f 1 0
f 1 1
f 1 2
f 1 3
f 2 0
f 2 3
```

Then the correct output will be:

```
g 3 4
b 0 0
b 0 1
b 0 2
b 0 3
b 1 0
b 1 1
b 1 2
b 1 3
b 2 0
b 2 3
+---+
|***|
|***|
|***|
|***|
+---+
u 2 1
+---+
|***|
|**4|
|***|
|***|
+---+
u 2 2
+---+
|***|
|**4|
|**4|
|***|
+---+
f 0 0
+---+
|f**|
|**4|
|**4|
|***|
+---+
f 0 1
+---+
|f**|
|f*4|
|**4|
|***|
+---+
f 0 2
+---+
|f**|
|f*4|
|f*4|
|***|
+---+
f 0 3
+---+
|f**|
|f*4|
|f*4|
|f**|
+---+
f 1 0
+---+
|ff*|
|f*4|
|f*4|
|f**|
+---+
f 1 1
+---+
|ff*|
|ff4|
|f*4|
|f**|
+---+
f 1 2
+---+
|ff*|
|ff4|
|ff4|
|f**|
+---+
f 1 3
+---+
|ff*|
|ff4|
|ff4|
|ff*|
+---+
f 2 0
+---+
|fff|
|ff4|
|ff4|
|ff*|
+---+
f 2 3
+---+
|fff|
|ff4|
|ff4|
|fff|
+---+
won
```

Writing your own test cases

Since your assignment will be automatically marked, it is crucial that you follow our instructions carefully. Your output will need to be in exactly the right format. To assist with this, we have made available some sample test cases. If you are logged in to the `ucpu[01]` machines, you can run these tests using the Makefile we provided by typing `make test`. If your output is incorrect, you will see both the expected output and your output. If you passed the test, then you will only see the name of the test.

The sample test cases are by no means exhaustive. You will need to test your code more thoroughly by thinking carefully about the specifications and writing your own tests.