THE UNIVERSITY OF SYDNEY

COMP2129 Assignment 1

Due: 9:00am Monday, 4 April 2016

This assignment is worth 8% of your final assessment

Task description

In this assignment you will develop a variation of the Minesweeper game in the C programming language. We recommend that you play the game to gain an overall understanding of how it works.

In each game, mines are hidden in a two dimensional mine field and the object of the game is to uncover the empty cells and flag the cells containing mines. To simplify your implementation, regions without mines should not be automatically expanded and players are not allowed to unflag cells once they have been flagged. Each game will contain a total of 10 mines, regardless of the grid size.

You are encouraged to ask questions on Ed using the assignments category. As with any assignment, make sure that your work is your own, and you do not share your code or solutions with other students.

Working on your assignment

You can work on this assignment on your own computer or the lab machines. It is important that you continually back up your assignment files onto your own machine, external drives, and in the cloud.

You are encouraged to submit your assignment on Ed while you are in the process of completing it. By submitting you will obtain some feedback of your progress on the sample test cases provided.

Academic declaration

By submitting this assignment you declare the following:

I declare that I have read and understood the University of Sydney Academic Dishonesty and Plagiarism in Coursework Policy, and except where specifically acknowledged, the work contained in this assignment or project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the the Academic Dishonesty and Plagiarism in Coursework Policy, can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Information Technologies, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and or communicate a copy of this assignment to a plagiarism checking service or in house computer program, and that a copy of the assignment may be maintained by the service or the School of IT for the purpose of future plagiarism checking.

Implementation details

Your program must be contained in one file called `minesweeper.c` and produce no errors when compiled using `clang` on the lab machines and Ed. Your program will be run from the command line using `./minesweeper` and read input from standard input and write output to standard output.

Important – your program will be marked automatically, so make sure that you follow the assignment specifications carefully. Your program output must match the exact output shown in the examples.

Program input

The program input consists of two sections. The first section defines the mine field and the second section describes the moves of the player. Each line of the input is one instruction. You are also required to check for whether any invalid input is encountered.

1. The first line of the input will be of the form `g <width> <height>` where `width` and `height` are integers in the range [1, 100]. This defines the size of the grid to `width` cells wide and `height` cells high. The grid must have at least 10 cells, so this instruction is only valid if `width` $\times$ `height` ≥ 10
2. The next 10 lines will be of the form `b <x> <y>` where `x` and `y` are integers in the range $[0, \texttt{width})$ and $[0, \texttt{height})$ respectively. This places a mine at cell (x, y) in the grid.
3. The next lines will describe the moves of the player – either `u <x> <y>` or `f <x> <y>`.
   - A `u` instruction means that the player uncovers the cell (x, y)
   - A `f` instruction means that the player flags the cell (x, y)

   There must be exactly one player instruction for each grid cell. That is, if the grid is of size `width` $\times$ `height` then there should be exactly `width` $\times$ `height` u and f instructions in total. No two instructions should refer to the same grid cell, and no more than 10 flags can be placed.

The input should be processed one line at a time. After reading a line of input your program should immediately act on the line of input as described before proceeding to the next line of input.

Sample input

In the following test case we construct a 4 x 4 mine field and then play the game.
The player successfully flags all of the mines and uncovers the remaining cells in this game.

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

Program output

1. After reading a `g <width> <height>` instruction, output the instruction that was inputted.
2. After reading a `b <x> <y>` instruction, output the instruction that was inputted. In addition, after reading the tenth `b <x> <y>` line, the state of the grid should be displayed.
3. After reading a correct player instruction, output the same instruction.
    - If the player uncovers a bomb, output the message `lost` and immediately exit.
    - If every mine has been flagged and every other empty cell has been uncovered, then display the final state of the grid, output the message `won` and then exit.
    - Otherwise, the state of the grid after the instruction should be outputted.
4. If an invalid, out of place, or otherwise incorrect instruction is read at any time, `error` should be outputted and your program should exit immediately. Make sure that you do not output the instruction or the grid following an erroneous instruction.

Displaying the grid

The grid is represented using 1 character per cell, with a 1 character border. A 4×4 grid looks like:

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

- The border consists of `-` or `|` characters, with a `+` character in each corner.
- The `*` character represents a covered cell.
- The `f` character represents a flagged cell.
- An uncovered cell is represented by a number that corresponds to the total number of bombs in the 8 adjacent cells that surround the uncovered cell.

The top left corner of the grid should have coordinates $(0, 0)$ and the bottom right corner of the grid should have coordinates $(\texttt{<width>} - 1, \texttt{<height>} - 1)$. The first coordinate is the `x` coordinate, and the second is the `y` coordinate. A 4×4 grid where the player has flagged $(3, 1)$ and uncovered $(0, 2)$ with 2 mines in the adjacent cells would look like:

```
+----+
|****|
|***f|
|2***|
|****|
+----+
```

Writing your own testcases

We have provided you with some sample testcases but these do not test all the functionality described in the assignment. It is important that you thoroughly test your code by writing your own testcases.

You should place all of your test cases in a `tests/` directory. Ensure that each test case has the `.in` input file along with a corresponding `.out` output file. We recommend that the names of your test cases are descriptive so that you know what each is testing, for example: `too-many-mines.in`

Submission details

Your attendance in the week 5 tutorial is required for the manual marking component.

You must submit your code in the assignment page on Ed. To submit, simply place your code into the workspace, then click the run button to check your program works and then click the submit button.

You are encouraged to submit multiple times, but only your last submission will be marked.

Marking

6 **marks** are assigned based on automatic tests for the *correctness* of your program.
This component will use our own hidden testcases that cover every aspect of the specification.

2 **marks** are assigned based on a manual inspection of the *style* and *quality* of your code and testcases.

Your program will be marked automatically, so make sure that you carefully follow the assignment specification. Your program output must match the exact output shown in the examples. You are encouraged to submit your assignment while you are working on it, so you can obtain some feedback.

Warning: Any attempts to deceive or disrupt the marking system will result in an immediate zero for the entire assignment. Negative marks can be assigned if you do not properly follow the assignment specification, or your code is unnecessarily or deliberately obfuscated.