INFO0004 - Project 2 (15%)
Power-11

Submission before December 2

You will be writing a C++ version of Power-11, a mathematical puzzle game. It is played on a squared n-by-n (n equals either 4 or 5) game board which begins with an initial configuration of two tiles, each of value 2 or 4, placed at random locations on the board (see fig. 1.a).

a)

```
Score: 0
#####################
#    |    |    |    #
#    |    |    |   4#
#   2|    |    |    #
#    |    |    |    #
#####################
```

b)

```
Score: 0
#####################
#    |    |    |    #
#    |    |    |    #
#    |    |    |    #
#   2|    |   2|   4#
#####################
```

c)

```
Score: 4
#####################
#    |    |    |    #
#    |    |   2|    #
#    |    |    |    #
#   4|   4|    |    #
#####################
```

d)

```
Score: 12
#####################
#    |    |    |    #
#   2|    |    |   2#
#    |    |    |    #
#   8|    |    |    #
#####################
```

Figure 1: *Example of a game, demonstrating a) initial setup, b) move 'down' and the creation of a new tile, c) move left, merging of two tiles and spawning a new tile, and d) move left, showing "motionless" merge and the creation of a new tile.*

The player then selects to move Up, Down, Left, or Right. When a direction of movement has been selected, all the tiles present on the board slide in that direction. "Sliding" means that each tile keeps moving until it either encounters a wall (edge of the board) or another tile (see fig. 1.b, resulting from selecting to move "down" from fig. 1.a). Note also that, after each move, a new tile with either value 2 or value 4, is placed randomly on

an open (empty) position of the board. In fig. 1.b, a tile with value 2 was (randomly) introduced on the fourth row, third column.

When 2 sliding tiles with the same value encounter each other, something interesting happens: they merge. When 2 tiles merge, they become a single tile whose value is the sum of the merged tiles. You can see merging in action when the user selects to move "left" from the configuration in fig. 1.b, yielding the board depicted in fig. 1.c (including the randomly generated tile on the second row and third column).

Note that merging does not disrupt sliding: at the end of a round, there is never any "holes" left on the board between tiles, in the direction of movement. Note also that only tiles that are already present on the board at the beginning of the round can merge.

Finally, tiles do not need to physically move to merge: if the user chooses to move left from the position on fig. 1.c, the resulting board is depicted in fig. 1.d (with a randomly generated tile on the second row, fourth column).

The game finishes when either of two conditions arises:

1) the board is full and there is no move that will trigger any tile merge. In this case, the user looses the game.

2) there is a tile whose value is 2^{11} on the board, in which case the user wins the game.

Throughout the game, the user accrues a score: starting at 0, the value of the tile resulting from a merge is added to the score whenever a merge occurs.

1 Creating the game

Your game will be played on the console. First, the game will start with a welcome screen, giving the users four options:

1) Option e/E: start a new easy (5-by-5) game;

2) Option h/H: start a new hard (4-by-4) game;

3) Option r/R: resume the last saved game;

4) Option q/Q: quit the game.

The last saved game must be in a file named `saved_game.txt` in the same directory as the game executable. The format of this file is:

- A line containing the score as a number.
- The game board stored one row per line. The board is stored in a properly framed format (with '`#`') and with columns delimited by '`|`'. In other words, the game state of fig.1.c would be saved as:

```
4
####################
#    |    |    |    #
#    |    |   2|    #
#    |    |    |    #
#   4|   4|    |    #
####################
```

Once a game has been started, the current state of the game is displayed:

- Whatever "title" you see fit;
- The text `Score:` followed by the current score is printed on a line;
- the board is then printed;
- the user is given the following options:
  - option u/U: move up;
  - option d/D: move down;
  - option l/L: move left;
  - option r/R: move right;
  - option s/S: save current game state to `save_game.txt`;
  - option q/Q: quit current game. This option must bring the user back to the welcome screen.

If the selected direction changes the state of the game[1], a randomly chosen open tile is given a value of 2 or 4, with probability 90% and 10% respectively.

As soon as the game is finished, the game state is printed with either:

- `YOU LOST!`
- `YOU WON WITH A SCORE OF` , followed by the final score.

Any input following the end of the game must bring the user back to the welcome screen.

[1] A selected direction of Down in fig. 1.b would not change the state of the game, and would therefore produce no effect on the game.

2 Bonus

For a bonus of 3 points, you can implement an "undo" facility in the game. The corresponding option should be "b/B" (for "Back"). This undo facility must allow the user to undo any series of move, all the way back to the start of the game.

Note that to function correctly across saved games, this facility must then ensure that the `saved_game.txt` file contains the series of game states encountered in the game.

3 Readability

Your code must be readable. Use indentation, one of the common naming conventions for variable names, and comments.

4 Robustness

Your code must be robust. The const keyword must be used correctly, sensitive variables must be correctly protected, memory must be managed appropriately, the programme must run to completion without crash.

5 Evaluation

This project counts towards 15% of your final mark. In this project, you will be evaluated on: readability of your code, organization of your code (including support for incremental compilation), usage of the STL, correctness, object-oriented design, judicious and correct use of C++ language features, code reusability, and efficiency of the solution.

6 Submission

Projects must be submitted before Wednesday December 2, 23:59 pm. After this time, a penalty will be applied to late submissions. This penalty is calculated as a deduction of $2^N - 1$ marks (where N is the number of started days after the deadline). You will submit your code (all appropriate files) and a `Makefile` (supporting incremental compilation) through the automated submission system. Your executable should be named `power`.

Basic tests will be automatically run on your submissions and feedback will be automatically sent to you. This feedback is for information only and does not necessarily reflect your final mark, nor does it wave the need for thorough testing. You can submit as often as you wish until the deadline. Submission instructions will be made available in due course, both via myULg and the web page of the course.

Your programme must compile on the ms8** computers, without error or warning. Failure to compile will result in an awarded mark of 0. Warnings systematically result in lost marks.

Have fun...