CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

C PROGRAMMING

Assignment 2: Minesweeper – A Variant Version of the Game.

BACKGROUND.

Minesweeper is a one-player computer game created by Curt Johnson and that was included in the Microsoft OS in the 90s. The goal of the game is to uncover all the squares that do not contain mines without being "blown up" by clicking on a square with a mine underneath.
- For more information on the game: https://en.wikipedia.org/wiki/Microsoft_Minesweeper
- For giving it a try: http://minesweeperonline.com/

In this assignment we are going to program a two-players variant of the game. On it, rather than avoiding the mines, each player will try to guess the locations of its oponent mines. First player uncovering all oponent’s mines wins the game.

GAME LOGIC.

To represent the information of the game we create a new datatype struct _game (which we simply type alias as game) and we create a single variable game g (which is the main actor in this story, as it is the one in which we store all the relevant info).

```
//-----------------------------------
// Datatype game
//-----------------------------------
struct _game {
   int mode;
   int status;
   int num_mines;
   int board_size;
   char* p1_name;
   char* p2_name;
   int p1_mines_uncovered;
   int p2_mines_uncovered;
   board_cell** p1_board;
   board_cell** p2_board;
};
typedef struct _game game;
```

The application contains a single game variable, which is created in the heap using malloc. Thus, the game variable is handled via a pointer located in the stack → game* g.

The fields (or pieces of information that we want to represent) of the variable game are:

1. **int mode** → Game mode being played. There are 4 possibilities:
   - mode = 0 → Player 1: Human; Player 2: Human
   - mode = 1 → Player 1: Human; Player 2: Computer
   - mode = 2 → Player 1: Computer; Player 2: Human
   - mode = 3 → Player 1: Computer; Player 2: Computer

   As expected, the human users will play the game by entering movements by keyboard. On the other hand, the computer will simply pick cells for its movements randomly.

2. **int status** → What is happening at the game now. There are 4 possibilities:
   - status = 1 → Game is on. Player 1 moves next.
   - status = 2 → Game is on. Player 2 moves next.
   - status = 3 → Game is over. Player 1 has won.
   - status = 4 → Game is over. Player 2 has won.

3. **num_mines** → The amount of mines to be allocated to each player.
4. **board_size** → The size of the board of each player.
5. **char* p1_name** → Name of Player 1.
6. **char* p2_name** → Name of Player 2.
7. **int p1_mines_uncovered** → Remaining mines p1 is to uncover to win the game.

8. **int p2_mines_uncovered** → Remaining mines p2 is to uncover to win the game.
9. **board_cell** p1_board** → Two dimensional array (board) of p1.
10. **board_cell** p2_board** → Two dimensional array (board) of p2.

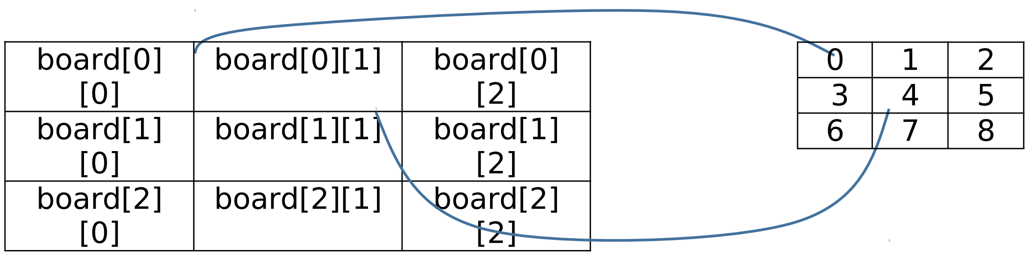
Regarding the latter p1_board and p2_board, please note we have defined them as board_cell** instead of board_cell[board_size][board_size]. The reason is to make the code compatible both with the gcc and cl compilers, as well as with C11 and C99 versions of the C language. In any case, for you as a programmer, you can think of them as two-dimensional arrays and thus access to each cell as p1_board[row][column] and p1_board[row][column]:

board[0][0]	board[0][1]	board[0][2]
board[1][0]	board[1][1]	board[1][2]
board[2][0]	board[2][1]	board[2][2]

To ease the interactive session with the user we identify each cell on the board with an integer value pos.
Thus, board[0][0] ⇔ pos = 0, board[0][1] ⇔ pos = 1, …, board[2][2] ⇔ pos = 8.

The mapping between both is very simple:

- Given a pos = k, it refers to board[k / board_size][k % board_size]
- Given board[row][column], it will take integer pos = (row * board_size) + column



board[0][0]	board[0][1]	board[0][2]
board[1][0]	board[1][1]	board[1][2]
board[2][0]	board[2][1]	board[2][2]

0	1	2
3	4	5
6	7	8

Also, the datatype struct _board_cell (simply aliased as board_cell) is used as C does not support tuples:

```
//-----------------------------------
// Datatype board_cell
//-----------------------------------
struct _board_cell {
   int value;
   boolean visible;
};
typedef struct _board_cell board_cell;
```

- The first component of the tuple, value, is an integer representing the number of surrounding mines (0 to 8) or taking the value 9 if the proper cell contains a mine.
- The second compoenent of the tuple, visible, is a boolean representing whether the value of the cell is to be displayed when showing the board (as your oponent has uncovered the cell before) or not (your oponent has not uncovered the cell yet).

INTERACTIVE SESSION.

The A02 folder includes the executable file A02.exe (Windows) and main_exec (Linux) with the application solved for you to try it.

Compare the executable your code generates to the aforementioned executable file, so as to ensure you are achieving the desired functionality. Also, to become familiar with the application itself and its parameters open a command line prompt and execute the program with the following configurations:

- A02.exe Jack Mary 3 25
  In this case, the two players (Jack and Mary) are humans and thus controlled by the user. The first name (in this case Jack) is Player1 and thus starts the game moving. The players will have a board of size 3x3, with the 25% of the cells being mines.
  The two players' goal is to win the game by uncovering the oponent's mines.

- A02.exe Computer Mary 4 10
  In this case, the first player is named "Computer", so it is controlled by the machine.
  As mentioned before, the machine will make its movements by picking randomly a cell. In this case, the two players will have a board of size 4x4, with the 10% of the cells being mines. The two players' goal is again to win the game by uncovering the oponent's mines.

- ex4.exe Computer Computer 3 25
  As you see, the case in which the computer plays against itself is supported as well.

FILES.

1. main.c

This is the file containing the main entry point of the project. You **do not** need to modify it.

The `int main(int argc, char* argv[])` just parses the 4 arguments from the program and calls to the function `play_game`. For debugging purposes, comment the lines parsing each of the parameters and force them to take concrete values. Once you have successfully programmed the application, you can uncomment the parameter parsing lines of code.

2. mine_ship_game.h

This is an external resource interface imported from 'main.c'. You **do not** need to modify it.

It specifies the datatypes `game` , `board_cell` and `boolean`.

Moreover, it specifies the following set of functions:

1. Function:

```
game* create_new_game(char* p1, char* p2, int board_size, int perc_mines);
```

It creates a new game provided the names of the two players, the size of the board and the percentage of cells in the board that are to be mines.
Note: This function is already implemented, you do not need to modify it.

2. Function:

```
int gen_num(int lb, int ub);
```

It returns a random number from [lb, ub). For example, if called gen_num(0,4) returns a random number from 0 to 3.
Note: This function is already implemented, you do not need to modify it.

3. Function:

```
char my_get_char();
```

It returns the keyboard character entered by the user.
Note: This function is already implemented, you do not need to modify it.

10. Function:

```
void play_game(char* p1, char* p2, int board_size, int percentage_mines);
```

This function puts together all the pieces.
It creates the variable game* g and performs the main loop of the game, displaying game status, asking for a new movement and performing it until the game is over.
Note: This function is already implemented, you do not need to modify it.

4. Function:

void place_mines_on_board(board_cell** board, **int** board_size, **int** num_mines);

It randomly allocates num_mines on the board board (setting these mine cells to value 9). It also populates the value of the remaining cells, by setting them to value = num of neighbour cells being mines.
Note: You have to implement this function.

5. Function:

void display_board_content(board_cell** board, **int** board_size, **int** mines_uncovered);
It prints the board, displaying only the cells that are visible.
Note: You have to implement this function.

6. Function:

int user_get_movement_index(**int** status, **int** board_size, board_cell** p1_board,board_cell** p2_board);
It gets the position of the next user movement.
Note: You have to implement this function.

7. Function:

int computer_get_movement_index(**int** status, **int** board_size, board_cell** p1_board,board_cell** p2_board);
It gets the position of the next computer movement.
Note: You have to implement this function.

8. Function:

int get_next_movement_index(**int** status, **char*** p1_name, **char*** p2_name, **int** board_size, board_cell** p1_board, board_cell** p2_board);
It gets the position of the player which, depending on status, will be the user or the computer.
Note: You have to implement this function.

9. Function:

void process_movement(**int** position_to_uncover, **int*** status, **int** board_size, board_cell** p1_board, board_cell** p2_board, **int*** p1_mines_uncovered, **int*** p2_mines_uncovered);
It processes the position_to_uncover movement selected by the current user.
Note: You have to implement this function.

3. mine_ship_game.c

This is the file implementing all the functions of game.h. You **have to** implement it.

MARK BREAKDOWN.

Assignment 2: 35 marks.

place_mines_on_board:	6 marks.
display_board_content:	6 marks.
user_get_movement_index:	6 marks.
computer_get_movement_index:	6 marks.
get_next_movement_index:	5 marks
process_movement:	6 marks.

To evaluate each function I will run it over some tests. The results are based on the performance of your code over these tests.

For each function, there are 5 possible scenarios:

A. The function passes all the test → 100% of marks.
B. The function does not pass all tests by small details → 70% of marks.
C. The function does not pass all tests by big details, but it compile and does not generate an exception (make the program crash) → 35% of marks.
D. The function was barely attempted, it does not compile or it generates and exception (makes the program crash) → 15% of marks.
E. The function was not attempted → 0% of marks.

SUBMISSION DETAILS.

Deadline.

Tuesday 29th of April, 11:59pm.

Submission Details.

Please submit to Canvas (folder 6. Submissions, A02) the following file:

- File “mine_ship_game.c”.

Lab Demo.

A brief individual interview about the assignment will take place on our lab session on Week 12. The demo is mandatory for the assignment to be evaluated.