CSCE 1030 – Homework 6

Due: 11:59 PM on Monday, May 4, 2015 CDT

Problem Statement:

For this C++ program, you will edit your Homework 5 programming assignment to add the functionality to play a simplified version of the classic Minesweeper game that will display to the screen. Unless so indicated, all of the requirements in Homework 5 hold in Homework 6. Note that there are some changes to this assignment!

You will take your solution from Homework 5 (or the solution posted on Blackboard, when available), and edit it as follows:

1. You shall organize your program into three files:
   - **`prgm.h`** will hold the include directives for the necessary libraries, any global constants, any enumerated data types, any type definitions, any structure definitions, and the list of function prototypes (i.e., function declarations).
   - **`main.cpp`** will hold the local include directive for the header file as well as the `main` function for the program.
   - **`func.cpp`** will hold the local include directive for the header file as well as all function definitions (not including `main`, of course) used in the program.
2. Define a structure that contains row-column coordinate of the game board. Specifically, this structure will contain the following two members: (1) an integer member that holds the row position on the board, and (2) an integer member that holds the column position on the board.
3. Update the function to display the game board by adding another parameter used to indicate whether or not to "reveal" the solution with all mines visible, or to display the current, active board with mines hidden (showing the initial value `'*'` instead). For squares containing a mine (i.e., `'@'`), if the boolean to reveal the board is set to `false`, you are to display the `'*'` character instead of `'@'` so as not to prematurely reveal the location of any of the mines. If the boolean is set to `true`, you will display either an `'X'` or a `'@'` for squares containing a mine, as appropriate. All other squares should be displayed as is (regarding the characters they represent based on their `enum` value. You will call this function every time to display the updated board after each valid user selection.
4. Add a boolean value-returning function to check the status of the coordinates selected by the user so as to return a boolean to indicate whether or not a mine was hit (i.e., if the user selected a square containing a mine, this function would return `true`; otherwise, it would return `false`). At a minimum, you will pass the game board as well as the structure variable containing the row-column coordinate described in #2 above to this function. You must use this structure variable to check the rows and columns of the board. If the square containing a mine is selected, you will set the value of the square to the `enum` representing that the user selected a square containing a mine (i.e., the `'X'`). If a previously selected square is selected, you are to indicate that an invalid position was entered and return `false`. Note that invalid selections (such as this case) should

not increment the counter for the number of turns. For other selections (i.e., squares containing the `enum` value representing the `'*'`), you will check all of the immediately adjacent squares on the board to see If they contain a mine and then you will write to the square the `enum` value representing the number of immediately adjacent squares containing mines. Note that immediately adjacent means the up-to-8 squares surrounding the aforementioned selected square. Squares on the edge of the board will not have eight immediately adjacent squares, so be sure to handle these cases.

5. Inside the `main` function, you will add a loop to play the game until either the game is won (user successfully selected 10 squares without hitting a mine) or lost (user selected a square containing a mine prior to selecting 10 squares). For each turn, the user will enter a set of coordinates corresponding to the position (i.e., row and column) on the board of the square the user wants to select. If the user enters an invalid coordinate position, you will indicate that an invalid position was entered and prompt the user to enter the coordinates again without incrementing the number of turns taken. You may assume that the user enters the position coordinates correctly as integers, though one of both values may not be in the valid range defined. Following each turn, you will display an updated version of the board. Ultimately, the game will be over if the user selects a square containing a mine (i.e., did not win the game) or the user exhausts all 10 turns without selecting a square containing a mine (i.e., won the game). At the end of the game, you will display a meaningful message about whether the user won or lost the game as well as the final updated board with the position of all the mines revealed.
6. Instead of a traditional two-dimensional array as was done in Homework 5, you shall declare and create a two-dimensional *dynamic* array in `main` to represent the 5-by-5 board for the `enum` type you declared to represent the various values that a square can assume. Since the board array is not global, you will pass the board array to the needed functions as a pointer (actually, as a pointer to a pointer). You will need to make sure to return the memory for the two-dimensional array back to the freestore when your program is done using it (at the end).

You may assume that all input by the user is of the correct data type, though perhaps out of range.

Extra Credit Opportunity: Up To 10 Points

For students who have completed all requirements for this program, you may add the following option to gain additional bonus points to your program score:

- You may add the option to "save" the game (perhaps using a sentinel value instead of the coordinates) so that it can be resumed at a later time. This would entail saving the game board (with all statuses indicated) and the number of turns the user has completed. This Information may be written to a file in any format you choose. Then when the program is run again, you can check If the user wants to resume the previous game and, if so, open the file (if it exists) and load the data into memory to continue the game.

Please note that unless you have completed all other requirements for this program no credit will be given for trying this extra credit. In other words, make sure your program is complete before attempting this extra credit.

Design:

On a piece of paper (or word processor), write down the algorithm, or sequence of steps, you will use to solve the problem. You may think of this as a "recipe" for someone else to follow. Continue to refine your "recipe" until it is clear and deterministically solves the problem. Be sure to include the steps for prompting for input, performing calculations, and displaying output.

Type these steps into a document (Word, txt, PDF, etc.). Note that this should be done before you start coding as completing it afterwards does not help you in learning the design process.

Implementation:

Now that you have a working design, your next step is to translate these steps into C++ code. Use the algorithm development techniques discussed in class to implement your solution to the problem above. Add your C++ code a little at a time, and compile and test as you go.

Remember to add your comments to your code to explain your program. Do this before/during programming instead of waiting until the end. At a minimum, you should comment the header (e.g., name, class, date, brief description of the program, etc.), all variables (i.e., what they are used for), and specific "blocks" of code. For example, use comments to describe the inputs, the formulas used, and any other important steps, such as loops, in your code.

Your program will be graded based largely upon whether it works correctly on a CSE Department machine, so you should make sure your program compiles and runs on a CSE machine.

Your program will also be graded based upon your programming style. At the very least, your program should include:

- A consistent indentation style as recommended in the textbook and in class;
- Meaningful variable names;
- A block header comment section that includes: your name, e-mail address, and a brief description of the program.

Testing:

Test your program to check that it operates as desired with a variety of inputs to make sure that all "paths" through you code are correct. Sample input and output appears below (with input shown in **bold**):

```
     +----------------------------------------------+
     |       Computer Science and Engineering       |
     |        CSCE 1030 - Computer Science I        |
     |   Student Name     EUID    euid@my.unt.edu   |
     +----------------------------------------------+

           Welcome to Minesweeper!

Enter number of mines to place on board (5 - 10): 6

----------------------------------------------------------
This computer program will randomly assign  6 mines to the
5 by 5 board. Your objective will be to select ten squares
on the board that do not contain mines using the given in-
formation from the adjacent squares. The game is over when
you either select 10 squares without hitting a mine or you
select a square containing a mine.
----------------------------------------------------------

Initializing board...assigning mines...now let's begin...

Enter position for move #1 (row[0-4] col[0-4]): 0 4
    0 1 2 3 4
  +-----------+
 0| * * * * 0 |
 1| * * * * * |
 2| * * * * * |
 3| * * * * * |
 4| * * * * * |
  +-----------+
Enter position for move #2 (row[0-4] col[0-4]): 0 3
    0 1 2 3 4
  +-----------+
 0| * * * 1 0 |
 1| * * * * * |
 2| * * * * * |
 3| * * * * * |
 4| * * * * * |
  +-----------+
Enter position for move #3 (row[0-4] col[0-4]): 1 3
    0 1 2 3 4
  +-----------+
 0| * * * 1 0 |
  1| *  *  *  3  * |
  2| *  *  *  *  * |
  3| *  *  *  *  * |
  4| *  *  *  *  * |
   +------------+
Enter position for move #4 (row[0-4] col[0-4]): 1 4
    0  1  2  3  4
   +------------+
  0| *  *  *  1  0 |
  1| *  *  *  3  1 |
  2| *  *  *  *  * |
  3| *  *  *  *  * |
  4| *  *  *  *  * |
   +------------+
Enter position for move #5 (row[0-4] col[0-4]): 4 2
    0  1  2  3  4
   +------------+
  0| *  *  *  1  0 |
  1| *  *  *  3  1 |
  2| *  *  *  *  * |
  3| *  *  *  *  * |
  4| *  *  1  *  * |
   +------------+
Enter position for move #6 (row[0-4] col[0-4]): 3 1
    0  1  2  3  4
   +------------+
  0| *  *  *  1  0 |
  1| *  *  *  3  1 |
  2| *  *  *  *  * |
  3| *  3  *  *  * |
  4| *  *  1  *  * |
   +------------+
Enter position for move #7 (row[0-4] col[0-4]): 4 4
    0  1  2  3  4
   +------------+
  0| *  *  *  1  0 |
  1| *  *  *  3  1 |
  2| *  *  *  *  * |
  3| *  3  *  *  * |
  4| *  *  1  *  0 |
   +------------+
Enter position for move #8 (row[0-4] col[0-4]): 4 3
    0  1  2  3  4
   +------------+
  0| *  *  *  1  0 |
  1| *  *  *  3  1 |
  2| *  *  *  *  * |
  3| *  3  *  *  * |
  4| *  *  1  0  0 |
   +------------+
Enter position for move #9 (row[0-4] col[0-4]): 3 4
    0 1 2 3 4
  +-----------+
 0| * * * 1 0 |
 1| * * * 3 1 |
 2| * * * * * |
 3| * 3 * * 1 |
 4| * * 1 0 0 |
  +-----------+
Enter position for move #10 (row[0-4] col[0-4]): 3 3
    0 1 2 3 4
  +-----------+
 0| * * * 1 0 |
 1| * * * 3 1 |
 2| * * * * * |
 3| * 3 * 2 1 |
 4| * * 1 0 0 |
  +-----------+
Congratulations! No mine hit in 10 tries!
    0 1 2 3 4
  +-----------+
 0| * @ * 1 0 |
 1| * * @ 3 1 |
 2| @ * @ * @ |
 3| * 3 * 2 1 |
 4| * @ 1 0 0 |
  +-----------+
```

Here is more sample input and output, this time for an unsuccessful attempt:

```
     +-----------------------------------------------+
     |       Computer Science and Engineering        |
     |        CSCE 1030 - Computer Science I         |
     |   Student Name     EUID    euid@my.unt.edu    |
     +-----------------------------------------------+

           Welcome to Minesweeper!

Enter number of mines to place on board (5 - 10): 4
Enter number of mines to place on board (5 - 10): 14
Enter number of mines to place on board (5 - 10): 8

----------------------------------------------------------
This computer program will randomly assign  8 mines to the
5 by 5 board. Your objective will be to select ten squares
on the board that do not contain mines using the given in-
formation from the adjacent squares. The game is over when
you either select 10 squares without hitting a mine or you
select a square containing a mine.
```

```
----------------------------------------------------------

Initializing board...assigning mines...now let's begin...

Enter position for move #1 (row[0-4] col[0-4]): 0 3
    0 1 2 3 4
  +-----------+
 0| * * * 3 * |
 1| * * * * * |
 2| * * * * * |
 3| * * * * * |
 4| * * * * * |
  +-----------+
Enter position for move #2 (row[0-4] col[0-4]): 5 2
Invalid position entered. Try again...
Enter position for move #2 (row[0-4] col[0-4]): 0 3
Invalid position entered (already used). Try again...
    0 1 2 3 4
  +-----------+
 0| * * * 3 * |
 1| * * * * * |
 2| * * * * * |
 3| * * * * * |
 4| * * * * * |
  +-----------+
Enter position for move #2 (row[0-4] col[0-4]): 4 1
    0 1 2 3 4
  +-----------+
 0| * * * 3 * |
 1| * * * * * |
 2| * * * * * |
 3| * * * * * |
 4| * 2 * * * |
  +-----------+
Enter position for move #3 (row[0-4] col[0-4]): 2 3
    0 1 2 3 4
  +-----------+
 0| * * * 3 * |
 1| * * * * * |
 2| * * * 4 * |
 3| * * * * * |
 4| * 2 * * * |
  +-----------+
Enter position for move #4 (row[0-4] col[0-4]): 0 0
    0 1 2 3 4
  +-----------+
 0| 1 * * 3 * |
 1| * * * * * |
 2| * * * 4 * |
 3| * * * * * |
 4| * 2 * * * |
  +-----------+
Enter position for move #5 (row[0-4] col[0-4]): 3 4
    0 1 2 3 4
  +-----------+
 0| 1 * * 3 * |
 1| * * * * * |
 2| * * * 4 * |
 3| * * * * 1 |
 4| * 2 * * * |
  +-----------+
Enter position for move #6 (row[0-4] col[0-4]): 6 8
Invalid position entered. Try again...
Enter position for move #6 (row[0-4] col[0-4]): 1 4
    0 1 2 3 4
  +-----------+
 0| 1 * * 3 * |
 1| * * * * X |
 2| * * * 4 * |
 3| * * * * 1 |
 4| * 2 * * * |
  +-----------+
Sorry, but you hit a mine. Tough break...
    0 1 2 3 4
  +-----------+
 0| 1 * @ 3 * |
 1| @ * * @ X |
 2| * * @ 4 @ |
 3| * @ * * 1 |
 4| @ 2 * * * |
  +-----------+
```

Documentation:

When you have completed your C++ program, write a short report (2 – 3 paragraphs) describing what the objectives were, what you did to solve the problem, and the status of the program. Does it work properly for all test cases? Are there any known problems? Also include a reflection in what you learned by completing this program that you anticipate taking forward as your knowledge in C++ programming grows.

Save this report in a separate file to be submitted electronically. You should also include any specific instructions required to compile or execute your code.

Homework Submission:

In this class, we will be using electronic homework submission to make sure that all students hand their programming projects (and labs) on time. You will submit your program source file to the class website through the "**Homework 6**" drop box by the due date and time.

Note that this project must be done individually. The program will be checked using a code plagiarism tool against other solutions, so please ensure that all work submitted is your own.

Note that the dates on your electronic submission will be used to verify that you met the due date above. All homework up to 24 hours late will receive a 50% grade penalty. Later submissions will receive zero credit, so hand in your best effort on the due date.

Summary:

- You will design an algorithm (or steps used) to solve the problem.
- You will implement your program on the CSE machines using C++. You will make sure to use good style, good variable names, indentation, etc. You will compile, run, and test your code.
- You will write a brief report describing what your code does and how well it works.
- You will submit electronically your C++ code, your design, and your brief report.

General Guidelines (for ALL of your programming assignments):

- Your program's output should initially display the department and course number, your name, your EUID, and your e-mail address.
- Use meaningful variable names.
- Use appropriate indentation.
- Use comments, including a program header. Example program header:

```
/*
 ============================================================================
 Name        : homework2.cpp
 Author      : Mark A. Thompson
 Version     :
 Copyright   : 2015
 Description : The program performs simple arithmetic operations based on in-
               put from the user.
 ============================================================================
 */
```

- Add a header to each function. Example function header:

```
/*
```

```
============================================================================
 Function    : deposit
 Parameters  : a double representing account balance and a double represent-
               ing the deposit amount
 Return      : a double representing account balance after the deposit
 Description : This function computes the account balance after a deposit.
============================================================================
 */
```