PIC 10A Section 2 - Homework #9 (due Monday, March 7, by 11:59 pm)

You should upload each .cpp (and .h file) separately and submit them to CCLE before the due date/time! Your work will otherwise not be considered for grading. Do not submit a zipped up folder or any other type of file besides .h and .cpp.

Be sure you upload files with the precise name that you used in your editor environment otherwise there may be linker and other errors when your homeworks are compiled on a different machine. Also be sure your code compiles and runs on Visual Studio 2022.

"LINESWEEPER"

This is kind of like a one-dimensional version of Mine Sweeper, if you ever played that game — it sometimes comes in pre-installed on Microsoft products.

The objective of this assignment is to get practice defining and implementing classes. In the game, the user provides their name, the number of locations they wish to play with, and the number of concealed mines. The user can then play as much as they like with these configurations.

Within the game, the user selects a position to reveal. If the location does not have a mine, they get 1 point; if the location has a mine, they get 0 points for the game; after a location has been revealed, the number of mines adjacent to that location (0, 1, or 2) is revealed to help guide the user's next move (assuming they don't blow themselves up with a mine).

They play until they either hit a mine (earning 0 points overall) or until they choose to no longer reveal locations, in which case they earn as many points as "safe" locations they have revealed. At the end of each game, the placement of all mines is revealed.

After each game, the user may be told they have a new high score if they in fact broke their prior record; otherwise they are told their overall statistics and they are asked if they want to play another game. See the example output for desired formatting.

In this homework, you will be writing several files...

- **Player.h**, a header file that defines the **Player** class along with
- **Player.cpp** that defines all of its member functions and constructor.
- You will also write **LineOfMines.h**, a header file that defines the **LineOfMines** class, along with
- **LineOfMines.cpp** that defines all of its member functions and constructor.
- You will also write and submit **Linesweeper.cpp** that runs the game through the main routine (one is provided to you that prints the instructions but nothing else — feel free to download it and edit it).

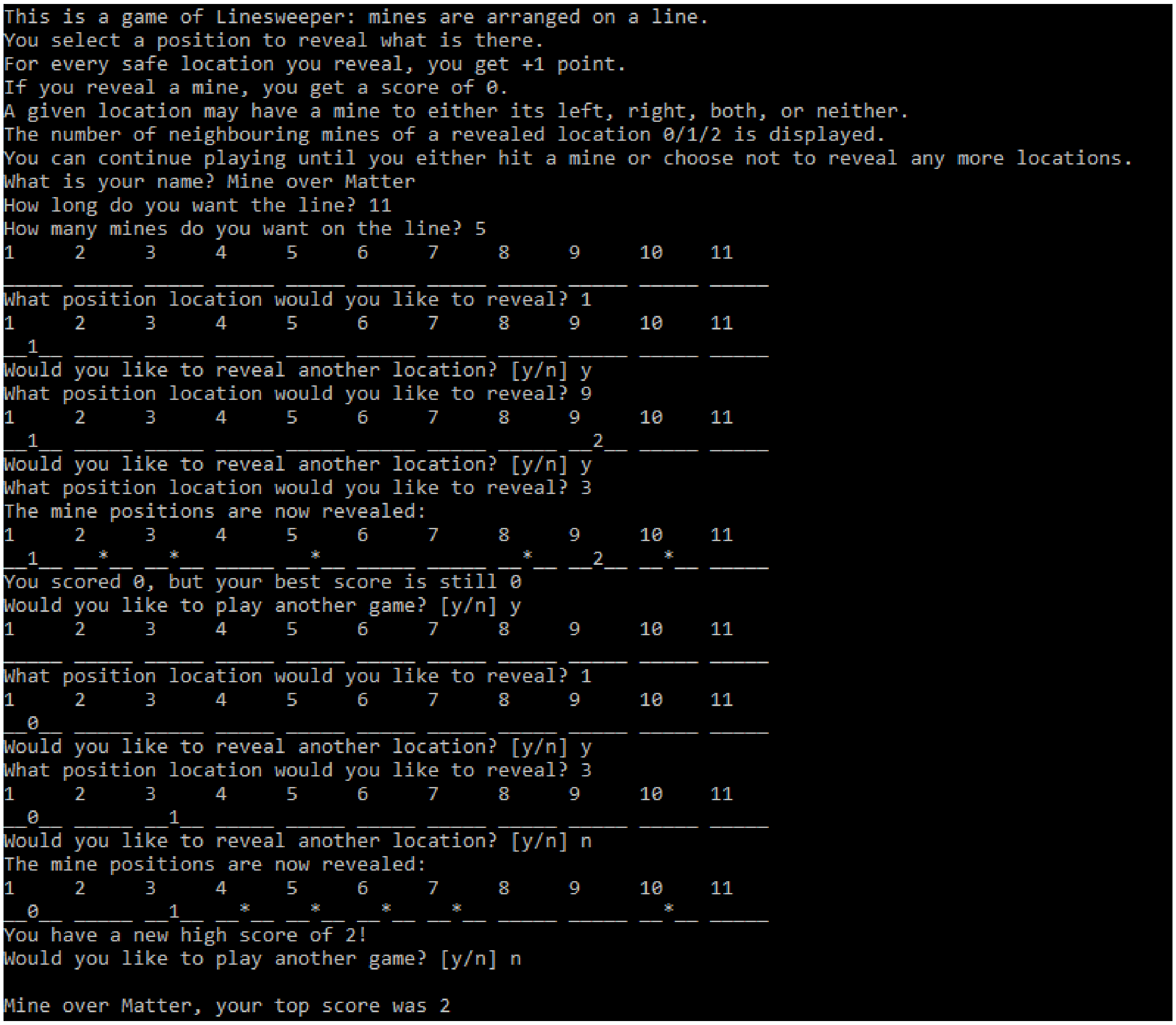
Thus, you will submit 5 files (not including the honesty statement) at the end of this homework! Be sure the files are named correctly!

Before discussing the class requirements, here is a sample output. The game begins by listing the rules (you can download "Linesweeper.cpp" with those instructions written for you). The user is then asked for their name, the line length, and how many mines to be used (you can assume that will never exceed the number of spaces per line).

After the initial prompts, the game begins by displaying the boarrd. The user is then repeatedly prompted with "What position would you like to reveal?" and "Would you like to reveal another location[y/n]?" to reveal locations and to determine if they are to continue playing (with 'y'). After each location they select, the line should then be displayed again to display the outcome of that choice.

Note that after each round (when a mine is hit or when the user declines to reveal more spots), if the player has a new high score they are told "You have a new high score of [NEW HIGH SCORE]!" And if their high score has not improved then they are told "You scored [POINTS JUST EARNED], but your best score is [OVERALL BEST SCORE]."

After every round, the user is asked "Would you like to play another game? [y/n]" and another round begins if they say 'y.' Otherwise, they are addressed with their name at the end with "[NAME], your top score was [OVERALL BEST SCORE]."



```
This is a game of Linesweeper: mines are arranged on a line.
You select a position to reveal what is there.
For every safe location you reveal, you get +1 point.
If you reveal a mine, you get a score of 0.
A given location may have a mine to either its left, right, both, or neither.
The number of neighbouring mines of a revealed location 0/1/2 is displayed.
You can continue playing until you either hit a mine or choose not to reveal any more locations.
What is your name? Mine over Matter
How long do you want the line? 11
How many mines do you want on the line? 5
1     2     3     4     5     6     7     8     9     10    11
_____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
What position location would you like to reveal? 1
1     2     3     4     5     6     7     8     9     10    11
__1__ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
Would you like to reveal another location? [y/n] y
What position location would you like to reveal? 9
1     2     3     4     5     6     7     8     9     10    11
__1__ _____ _____ _____ _____ _____ _____ _____ __2__ _____ _____
Would you like to reveal another location? [y/n] y
What position location would you like to reveal? 3
The mine positions are now revealed:
1     2     3     4     5     6     7     8     9     10    11
__1__ __*__ __*__ _____ __*__ _____ _____ __*__ __2__ __*__ _____
You scored 0, but your best score is still 0
Would you like to play another game? [y/n] y
1     2     3     4     5     6     7     8     9     10    11
_____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
What position location would you like to reveal? 1
1     2     3     4     5     6     7     8     9     10    11
__0__ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
Would you like to reveal another location? [y/n] y
What position location would you like to reveal? 3
1     2     3     4     5     6     7     8     9     10    11
__0__ _____ __1__ _____ _____ _____ _____ _____ _____ _____ _____
Would you like to reveal another location? [y/n] n
The mine positions are now revealed:
1     2     3     4     5     6     7     8     9     10    11
__0__ _____ __1__ __*__ __*__ __*__ __*__ _____ _____ __*__ _____
You have a new high score of 2!
Would you like to play another game? [y/n] n

Mine over Matter, your top score was 2
```

A **Player** class should

- have a **std::string** member variable, **name** storing the player's name.
- have an **int** member variable, **bestScore**, storing the player's best score achieved.
- have a constructor, that can be given a **std::string** input, assigning **name** to this input and **bestScore** to 0.
- have a **getName** function that returns the player's name.
- have a **getMaxScore** function that returns the player's best score up to that point.
- have a **checkRecord** function that accepts an **int** for a recent score, updating the maximum score if this input is greater; it should also return **true** to indicate the new score is *strictly* better, if it is, and otherwise return **false**.

A **LineOfMines** class should

- have two **size_t** variables, **lineLength**, to store the number of locations in the game, and **numberOfMines**, to store the number of mines.
- have an **int** member, **score**, to store the game's score.
- have a **std::vector<std::string>** member, **locationDisplays**, to store how each location should be displayed if revealed.
- have a **std::vector<int>** member, **neighbouringMineCounts**, to store the number of neighbouring mines for each position.
- have a **std::vector<int>** member, **mineLocations**, to store the locations of all mines in the game.
- have a **bool** member, **hitMine**, to store whether or not a mine has been hit.
- have a constructor that accepts two **size_t** inputs for the number of locations and number of mines, respectively and
  - sets **lineLength** and **numberOfMines** appropriately,
  - sets **score** to 0,
  - sets **locationDisplays** to a **std::vector<std::string>** of size **lineLength** all with value "_____" (5 underscores),
  - sets **neighbouringMineCounts** to a **std::vector<int>** of size **lineLength**, all with value 0,
  - sets **mineLocations** to a **std::vector<int>** of size **numberOfMines**, all with value 0,
  - sets **hitMine** to **false**,
  - and then places the mines in the game and sets the counts.
- a member function **placeMines**, to be called in the constructor body, to place **numberOfMines** mines in distinct locations from **1** to **lineLength**.
- a member function **setCounts**, to be called in the constructor body, to set the values of **neighbouringMineCounts** appropriately.
- a **containsMine** member function accepting an **int** for a position number (from **1** to **lineLength**), returning **true** if there is a mine there and otherwise false;
- a **display** member function that displays the board: it should output each position from **1** to **lineLength** on one line and each value of **locationDisplays** on the next line (all of which *must have a width of 5*, each separated by a space, such that the position numbers align on the left of their corresponding **locationDisplays** value.

- a **grandReveal** member function that modifies each **locationDisplays** value to **“__*__”** if there is a mine there, and subsequently calls **display**.
- a **hasHitMine** member function that returns whether or not a mine has been hit; **true** if yes, **false** if not.
- a **getScore** member function that returns the player’s score: 0 if they have hit a mine and otherwise the number of safe locations they have revealed.
- a **makeSelection** member function, accepting an **int** for the position a user wishes to reveal, updating the corresponding **locationDisplays** value to **“__*__”** if there is a mine present (and sets **hitMine** to **true** if there is a mine), and otherwise sets the value to one of __0__, __1__, or __2__, depending on how many adjacent mines there are.

Hints:

Do not worry about running the entire game to start with. Proceed as follows:

1. Write the **Player** class and make sure that you can use it in a **main.cpp** file that you write. This should be pretty easy to test since there are very few data and functions.

2. Write the **LineOfMines** class slowly, testing it along the way. If you follow these steps, it should be pretty simple. Do one step at a time where you write the declaration and definition only as needed at the given stage.

   (a) First, write the constructor, but do not use or make reference to **placeMines** or **setCounts**. Ensure your code compiles if you try to call the constructor.

   (b) Next, write **getScore** and make sure it works: at this point, it should return **0**.

   (c) Next, write **hasHitMine** and make sure it works: at this point, it should return **false (0)**.

   (d) Write **display** next. You may find the <**iomanip**> header useful for part of it. When you call display at this point, you should just see the position numbers on one line and each **locationDisplays** string should come out as "_____", properly aligned, on the next.

   (e) Write **placeMines** (the logic should be similar to the drawing from a deck of cards) and be sure to call it from your constructor.

   (f) Write **containsMine**.

   (g) Write **grandReveal** and test it. You should, for example, be able to see the mines displayed, and the number of mines present should match the number of mines you requested to be placed on the line.

   (h) Write **setCounts**: as you iterate through the positions, you will need different logic depending on whether you are at the very left of the line, very right of the line, or somewhere in the middle. Be sure to call this function from your constructor once written.

   (i) Write **makeSelection**.

3. Now you're ready to write a main routine to handle the full game logic.