Minesweeper is a game with a two-dimensional grid playing field. Each grid location (called a cell) may contain a mine. Initially, all cells are covered, hiding any mines from the player. Play is conducted as follows:

- A cell may be uncovered. If a mine is uncovered, the game is lost. Otherwise, the total number of mines in any adjacent grid cells (all 8 directions) is shown in the cell. Using these numbers, together with skill, judgment, and a sprinkling of luck, a player should be able to determine the locations of all mines.
- A covered cell may be marked. This is used to indicate a grid cell where the player thinks a mine is.
- A covered cell may be unmarked. This removes a mark from a grid cell.
- The game is won if all grid cells have been either marked or uncovered, and the number of marked grid cells is the same as the number of mines. I.e. the player has identified all the mines without uncovering.

Consider the following specification for a SweeperGrid class that represents the Minesweeper playing field. Each grid cell contains a SweeperCell object, that contains information regarding the state of the cell. (The SweeperCell class is provided and is explained below.) Cell locations are given as coordinates (row, col) where (0,0) is the upper left-hand corner of the grid.

**Specification for SweeperGrid Class**

**Data Attributes**

For this project, a SweeperGrid is modeled using a dynamically-allocated two-dimensional array of SweeperCells as demonstrated in lecture to allow grids of differing sizes to be created. Thus the data attributes include at least those shown below. You may add additional appropriate attributes.

<table>
<thead>
<tr>
<th>Objects</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of rows</td>
<td>int</td>
<td>numRows</td>
</tr>
<tr>
<td>number of columns</td>
<td>int</td>
<td>numColumns</td>
</tr>
<tr>
<td>number of bombs in the grid</td>
<td>int</td>
<td>numBombs</td>
</tr>
<tr>
<td>pointer to grid</td>
<td>SweeperCell**</td>
<td>grid</td>
</tr>
</tbody>
</table>

**Operations**

- Explicit-value constructor - receives the number of rows and columns, and a density percentage. Default
values are given below. Throws a RangeError exception if initialRows or initialCols is less than 5 (to make sure the grid is large enough) or if the density is not between 25 and 75 (to make sure not all cells are empty or all cells contain bombs).

It should dynamically allocate a two-dimensional SweeperCell grid of the specified size. The constructor places density percentage number of bombs in the grid, and computes the number of adjacent cells containing bombs for each cell.

Bombs are to be placed randomly into the grid in the density specified. Density is specified as an integer between 25 and 75 that represents the percentage of cells that will contain bombs. (E.g. 25 means 25%, or one-fourth, of the cells will contain bombs.) The easiest way to do this is to loop through the grid cells, and for each cell use the built-in random number generator to pick a number between 1 and 100. If the number generated is less than the density percentage, then place a bomb in the cell.

Note: the random number generator under g++ works as follows. The generator is seeded using srand() (defined in <cstdlib>), which receives a long integer. We want the game to be different each time, so most programmers use the result of the time function (defined in <ctime>) to seed the generator by using:

    srand(time(0));

This should be done once before the first use of the random number generator. (Note: if you want the same numbers to be generated each time, e.g., when testing, use a constant as the argument to srand(), but your final submission should use time()).

The function rand() returns the next random integer between 0 and RAND_MAX. Scaling this result to numbers between 1 and 100 can be done by the following code fragment:

    rand() % 100 + 1

Recall that % is the remainder operator in C++. (As shown in the sample run below, the actual density percentage produced by this method can be off by quite a bit, but it should be within 10% or so.)

Analysis

<table>
<thead>
<tr>
<th>Objects</th>
<th>Default</th>
<th>Type</th>
<th>Kind</th>
<th>Movement</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of rows</td>
<td>5</td>
<td>int</td>
<td>variable</td>
<td>received</td>
<td>initialRows</td>
</tr>
<tr>
<td>number of columns</td>
<td>5</td>
<td>int</td>
<td>variable</td>
<td>received</td>
<td>initialCols</td>
</tr>
<tr>
<td>density percentage</td>
<td>25</td>
<td>int</td>
<td>variable</td>
<td>received</td>
<td>density</td>
</tr>
</tbody>
</table>

- Copy constructor - creates a new SweeperGrid that is identical to an existing one.

Analysis
- Destructor - deallocates the grid
  Analysis - no objects

- operator= - overloaded assignment operator function. Makes an existing SweeperGrid object identical to the original SweeperGrid object.

  Analysis

<table>
<thead>
<tr>
<th>Objects</th>
<th>Type</th>
<th>Kind</th>
<th>Movement</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>original SweeperGrid object</td>
<td>SweeperGrid</td>
<td>variable</td>
<td>received</td>
<td>original</td>
</tr>
<tr>
<td>this SweeperGrid object</td>
<td>SweeperGrid</td>
<td>variable</td>
<td>returned</td>
<td>*this</td>
</tr>
</tbody>
</table>

- GetRows - returns the number of rows in the grid

  Analysis

<table>
<thead>
<tr>
<th>Objects</th>
<th>Type</th>
<th>Kind</th>
<th>Movement</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of rows</td>
<td>int</td>
<td>variable</td>
<td>returned</td>
<td>numRows</td>
</tr>
</tbody>
</table>

- GetColumns - returns the number of columns in the grid

  Analysis

<table>
<thead>
<tr>
<th>Objects</th>
<th>Type</th>
<th>Kind</th>
<th>Movement</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of columns</td>
<td>int</td>
<td>variable</td>
<td>returned</td>
<td>numColumns</td>
</tr>
</tbody>
</table>

- GetBombs - returns the number of bombs in the grid

  Analysis

<table>
<thead>
<tr>
<th>Objects</th>
<th>Type</th>
<th>Kind</th>
<th>Movement</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of bombs</td>
<td>int</td>
<td>variable</td>
<td>returned</td>
<td>numBombs</td>
</tr>
</tbody>
</table>

- GameWon - returns true if the game has been won

  Analysis

<table>
<thead>
<tr>
<th>Objects</th>
<th>Type</th>
<th>Kind</th>
<th>Movement</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>win result</td>
<td>bool</td>
<td>variable</td>
<td>returned</td>
<td>---</td>
</tr>
</tbody>
</table>

- PlaceBomb - place a bomb in the grid cell at location (row, col). Note this function must recalculate the number of adjacent bombs of the cell's neighbors (all 8 directions). If the location (row, col) is not within the bounds of the grid, throws a RangeError exception.
• RemoveBomb - remove a bomb in the grid cell at location (row, col). Note this function must recompute the number of adjacent bombs of the cell's neighbors (all 8 directions). If the location (row, col) is not within the bounds of the grid, throws a RangeError exception.

Analysis

<table>
<thead>
<tr>
<th>Objects</th>
<th>Type</th>
<th>Kind</th>
<th>Movement</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>row index</td>
<td>int</td>
<td>variable</td>
<td>received</td>
<td>row</td>
</tr>
<tr>
<td>column index</td>
<td>int</td>
<td>variable</td>
<td>received</td>
<td>col</td>
</tr>
</tbody>
</table>

• Uncover - uncovers the grid cell at location (row, col). Returns true if a bomb is uncovered. If the location (row, col) is not within the bounds of the grid, throws a RangeError exception.

Analysis

<table>
<thead>
<tr>
<th>Objects</th>
<th>Type</th>
<th>Kind</th>
<th>Movement</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>row index</td>
<td>int</td>
<td>variable</td>
<td>received</td>
<td>row</td>
</tr>
<tr>
<td>column index</td>
<td>int</td>
<td>variable</td>
<td>received</td>
<td>col</td>
</tr>
</tbody>
</table>

• Mark - marks the grid cell at location (row, col). If the location (row, col) is not within the bounds of the grid, throws a RangeError exception.

Analysis

<table>
<thead>
<tr>
<th>Objects</th>
<th>Type</th>
<th>Kind</th>
<th>Movement</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>row index</td>
<td>int</td>
<td>variable</td>
<td>received</td>
<td>row</td>
</tr>
<tr>
<td>column index</td>
<td>int</td>
<td>variable</td>
<td>received</td>
<td>col</td>
</tr>
</tbody>
</table>

• Unmark - unmarks the grid cell at location (row, col). If the location (row, col) is not within the bounds of the grid, throws a RangeError exception.

Analysis
Write - outputs the SweeperGrid in a two-dimensional grid to an output stream. (SweeperCell implements operator<< that outputs a cell as a single character. See SweeperCell implementation for details.) The columns of the grid should line up as shown in the sample run. (I.e., there should be two spaces in front of every cell and a newline should be after the last cell in each line.)

Analysis

Assignment

The SweeperCell class and the RangeError classes are provided for this assignment. They may be copied on the csserver using:

```
    cp /home/hwang/cs215/project4/*.* .
```

There are three files, except.h, sweepercell.h and sweepercell.cpp. You may not modify these files. However, if you feel you need more operations from the SweeperCell class, talk to the instructor. The SweeperCell class defines operations that allow the SweeperCell object to be manipulated and accessed in accordance with the Minesweeper game rules. The files contain comments explaining what each function does. Hopefully, this is sufficient.

Write the implementation of the SweeperGrid class specified above. The SweeperGrid class definition should be put in header file sweepergrid.h with suitable compilation guards. The implementations of the SweeperGrid member functions should be put in source file sweepergrid.cpp. The SweeperGrid class must be implemented using a dynamically-allocated two-dimensional array as discussed in lecture. Projects that do not use a dynamically-allocated two-dimensional array will be returned for resubmission with late penalty. The member function names and the order of the parameters must be as specified above. Your code will be linked with a grading driver program that expects this. Note that the main program will not be using all of the specified functions. However, all of the functions must be correct to receive full credit.

Write a main program sweeper.cpp that implements the Minesweeper game described above using the SweeperGrid class. This game will be text-based and interactive. It should have the following features:

- It should be able to construct a user-specified size game grid with a user-specified density of bombs by accepting the number of rows, the number of columns, and the density of bombs to be created as command-line arguments. I.e., there will be a total of 4 arguments on the command line. (See example run below.) The program should attempt to construct a SweeperGrid using these arguments. The program should handle any RangeError exceptions resulting from the construction by printing an error message and exiting the program.
Reminder: since `argv` is an array of C-strings, the command-line arguments will need to be converted to integers using the function `atoi` that is defined in `<cstdlib>`. (This was used in Homework 2.)

- After construction, the number of bombs in the grid should be output to the screen.
- The game must allow the user to repeatedly uncover a cell, mark a cell, or unmark a cell until a bomb is uncovered (and the game is lost), the game is won, or the player wants to quit. This most likely will be a menu-driven loop.
- The game grid should be output to the screen before every player move.
- When the game is over (by whatever means), the game grid with all cells uncovered should be output to the screen.
- In addition, the program must handle the player entering an illegal location (i.e., indexes not in bounds) by handling the RangeError exceptions thrown by the SweeperGrid functions. Such locations should be ignored by the program and the user asked to input a new location, or a new action choice and new location.

A sample run of a program meeting these specifications is shown below.

You must submit a makefile named `Makefile.project4` that creates an executable named `sweeper`. Submissions without working makefiles will be assessed up to a 3-point penalty as indicated in the syllabus. It should conform to the examples given in the handout `Very Basic make` and demonstrated in class.

REMINDER: Your project must compile for it to be graded. Submissions that do not compile will be returned for resubmission and assessed a late penalty. Submissions that do not substantially work also will be returned for resubmission and assessed a late penalty.

Follow the guidelines in the `C++ Programming Style Guideline` handout. As stated in the syllabus, part of the grade on a programming project depends on how well you adhere to the guidelines. The grader will look at your code listing and grade it according to the guidelines.

**What to submit**

Electronically submit a tarfile containing `Makefile.project4`, `sweepergrid.h`, `sweepergrid.cpp`, and `sweeper.cpp` as explained in the handout `Submission Instructions for CS 215`. Please do not submit `except.h`, `sweepercell.h`, `sweepercell.cpp`, object files, or executable files.

Turn in hardcopy printouts of `Makefile.project4`, `sweepergrid.h`, `sweepergrid.cpp`, and `sweeper.cpp`. Please do not turn in `except.h`, `sweepercell.h`, `sweepercell.cpp`. 

02/23/2010
Sample run

$ ./sweeper 8 12 25
There are 34 bombs in the grid.

# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #

Enter c to uncover, m to mark, u to unmark, q to quit: c
Enter a location (row col) to uncover: 0 0

2 # # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #

Enter c to uncover, m to mark, u to unmark, q to quit: c
Enter a location (row col) to uncover: 0 2

2 # 1 # # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #

Enter c to uncover, m to mark, u to unmark, q to quit: c
Enter a location (row col) to uncover: 0 3

2 # 1 0 # # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #

Enter c to uncover, m to mark, u to unmark, q to quit: c
Enter a location (row col) to uncover: 1 2

```
2 # 1 0 # # # # # # 
# # 1 # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
```

Enter c to uncover, m to mark, u to unmark, q to quit: c
Enter a location (row col) to uncover: 1 3

```
2 # 1 0 # # # # # # 
# # 1 1 # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
```

Enter c to uncover, m to mark, u to unmark, q to quit: c
Enter a location (row col) to uncover: 0 4

```
2 # 1 0 1 # # # # # # 
# # 1 1 # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
```

Enter c to uncover, m to mark, u to unmark, q to quit: c
Enter a location (row col) to uncover: 1 4

```
2 # 1 0 1 # # # # # # 
# # 1 1 2 # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
# # # # # # # # # # 
```

Enter c to uncover, m to mark, u to unmark, q to quit: c
Enter a location (row col) to uncover: 0 5

```
```
Enter c to uncover, m to mark, u to unmark, q to quit: m
Enter a location (row col) to mark: 1 5

Enter c to uncover, m to mark, u to unmark, q to quit: m
Enter a location (row col) to mark: 1 0

Enter c to uncover, m to mark, u to unmark, q to quit: c
Enter a location (row col) to uncover: 1 1
You've uncovered a mine! Game over!!
Extra credit

There are two additions that may be completed for extra credit. Note that you may want or need to add additional member functions or data members to the SweeperGrid class to support these additions. (But it is still the case that the SweeperCell class should not be modified.) If you do the extra credit, you are expected to submit two versions of the project, one regular project, and one with any extra credit attempts.

- (5 points) Implement the auto-open feature when uncovering a cell with no bombs adjacent to it. When such a cell is uncovered, the program automatically uncovers all adjacent cells, since this is safe to do. This process is repeated on any newly uncovered cell with no bombs adjacent to it. Note that this is similar to finding blobs in Project 2.
- (5 points) Implement the no bomb on first move feature where the first move a player makes will never uncover a bomb. There are a number of different ways to do this.