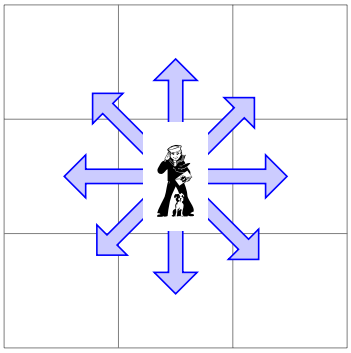
CSC 112 Lab 4
Safety Dance
Spring 2015

Due
3:00pm Friday, 2/20

1 Introduction

Consider a drunken sailor[1] staggering about a dance floor. Assume the sailor is initially located in the middle of the dance floor that is divided into a grid and can move in one of eight directions as seen in the diagram below. The sailor is looking for his date, but must avoid running into one of his ex's who are randomly located on the floor. Given the sailor must take a maximum number of steps, will he find his date before running into an ex or before he simply passes out?



1.1 Sailor's Dance Success or Fail

We are interested in determining the outcome of the sailor staggering about the dance floor. As done in Lab 3 (Drunken Sailor), simple method for determining the outcome is to simulate a staggering person. The program will prompt the user for the number of ex's, the maximum number of dance moves (steps), and a random seed. After the information is entered, your program will then run the experiment requested. The program will first randomly place the date and the specified ex's around the dance floor. For very move the sailor makes the program will clear the screen and graphically display the dance floor, which also includes the steps made by the sailor, and the locations of the ex's and date. At the end of the experiment, display the final dance floor, the result, and the number of steps made. An example of a 10×10 dance floor, 5 ex's, 80 maximum steps and 70 seed value given below. Note this is only an example, not necessarily what you should expect. In this example the sailor (*) found his date ($) in 67 dance moves!

Terminal

```
> ./lab4
Enter the number of ex's, max number of moves, and random seed -> 5 80 70
----------------------
|                      |
|        X             |
|        .      *$     |
|      . X     .       |
|.   .   .   .         |
|. :   :   : X         |
|. . @ .   X .         |
|. : .   . . X         |
|. . : .             . |
|. : . . .             |
----------------------
Found date at (2, 8) after 67 moves
```

[1] Of course the sailor is of age; regardless, we certainly don't condone alcohol. Booooo alcohol! It only leads to staggering.

2 Simulating the Dance Floor and Moves

Assume the dance floor is a 10×10 grid, where each square (2-D array element) is a possible location for the sailor. Therefore the sailor's location consists of a row and column location. The sailor will always start in the middle of the dance floor (5, 5). The safety dance of the sailor is simulated by randomly adding a -1 or +1 it to the current row position of the sailor, then repeating for the column location. Note the sailor is not allowed to step off the dance floor. This process (randomly updating the row and column locations) repeats until the sailor finds his date, finds an ex, or runs out of steps (passes out).

2.1 Modeling the Dance Floor and Location of People

As mentioned before, the dance floor is a 10×10 grid. This is best modeled as a 10×10 `int` matrix, where each value of the matrix stores the number of times the sailor has moved to that element (square). The matrix should also store the location of the ex's and the date. An easy way to store locations of these people is to set the corresponding location in the matrix to a negative number. For example, store the value -2 for ex's and the value -1 for the location of the date.

```
int main()
{
   int floor[MAX_SIZE][MAX_SIZE] = {};  ///> the dance floor yo
   int numEx;     ///> number of ex's
   int maxMoves; ///> beyond this you pass out
   int seed;     ///> getting more random up in here

   getExperimentData(numEx, maxMoves, seed);
   srand(seed);

   setupDanceFloor(numEx, floor);
   // more interesting code follows...
```

The function `setupDanceFloor` will set the locations of the date and the ex's. Note the values stored.

```
void setupDanceFloor(int numEx, int floor[][MAX_SIZE])
{
   int row;  ///> local variable for the current row
   int col;  ///> local variable for the current column

   // randomly place the date
   row = rand()%MAX_SIZE;
   col = rand()%MAX_SIZE;
   floor[row][col] = -1;    // -1 indicates the date

   // randomly place the ex's
   for(int i = 0; i < numEx; i++)
   {
      row = rand()%MAX_SIZE;
      col = rand()%MAX_SIZE;
      floor[row][col] = -2; // -2 indicates an ex
   }
}
```

2.2 Displaying the Dance Floor

As described in the introduction, your program must display the dance floor after every step of the sailor. A function for clearing the screen is available at the course web-site. The dance floor is a graphical representation of the current simulation state; therefore, do not print the numbers in your dance floor matrix. Instead, you will print a certain character for each value in the matrix.

The date should be indicated with a `'$'`, the ex's with a `'X'`, and the current location of the sailor is a `'*'`. In addition for every location the sailor has visited display a character corresponding to the number of times he has been there. If the sailor has never visited the square then print a space. If the sailor has visited a square only once, then print a period (`'.'`). If the sailor has visited a square no more than twice, then print a colon (`':'`). If the sailor has visited a square more than twice then print an at symbol (`'@'`). Note, print a space after each symbol to make the dance floor appear more square.

2.3 Displaying Success, Fail, or Meh

If the sailor finds his date, display a message, the location of the date, and how many moves made. Similarly, if the sailor finds an ex, display message, the location of the ex, and how many moves made. If the sailor makes the maximum number of moves yet finds no one, print a message indicating this and the number of moves made.

3 Programming Points

You **must** adhere to all of the following points to receive credit for this lab assignment.

1. Create a directory `Lab4` off of your `CSC112` directory to store your program in a file called `lab4.cpp`
2. Your program must be modular in design.
3. Your `main` function can only consist of variable declarations, function calls, and control structures (no input or output in the `main` function).
4. Your program must compile cleanly, no errors or warnings are allowed.
5. Pass variables appropriately. Only pass by reference when necessary.
6. Your program must adhere to documentation style and standards. Don't forget function headers and variable declarations.
7. **Turn-in** a print-out of your program source code (`lab4.cpp`). In addition, copy your program source code to your `Gade/Lab4` directory.