The Borough of Manhattan Community College
The City University of New York

SCI 120 Prof. Betancourt
Midterm Exam Date_____

Name: Last_____, First_____

WRITE A PROGRAM that will execute the indicated pseudo-code and reproduce the example.

- Write your code in the provided paper.
- Enter one character per box with a pencil
- The use scratch paper is recommended
- Use the provided C++ Quick Reference
- Write all syntax correctly for full credit

Description: **Minesweeper (game)**

Minesweeper is a logic puzzle video game. The game features a grid of clickable tiles, with hidden "mines" (depicted as naval mines in the original game) dispersed throughout the board. The objective is to clear the board without detonating any mines, with help from clues about the number of neighboring mines in each field. Cells have three states: unopened, opened and flagged. An unopened cell is blank and clickable, while an opened cell is exposed. Flagged cells are those marked by the player to indicate a potential mine location. If a player opens a mined cell, the game ends. Otherwise, the opened cell displays either a number, indicating the number of mines vertically, horizontally or diagonally adjacent to it, or a blank tile, and all adjacent non-mined cells will automatically be opened. The present program only builds and displays a board with mines placed at random and the count of adjacent mines. It does not includes the game play, scores, and fancy graphics.

Task: (This is a general description of the algorithm. Follow the pseudo code in the next page) Write a code that defines a `MineSweeper` structure that contains a string that hold the board contents, the number of rows and columns, and the mine character. Create a function to add a mine at random in an empty cell, with a bias to be close to other mines. This function should call another function that increments by one the count of mines of adjacent cells (including diagonals). One more function displays the board, showing the mines, the number of adjacent mines, empty if there is no adjacent mines, and lines and spaces to make the board more readable on the standard output.

Program INPUT: (hardwire)

```
The board size, number of mines, and mine symbol
```

Program OUTPUT:

```
The board shown in the standard output with the mine locations and number
of adjacent mines.
```

Run Example: (Shows mines * and adjacent mines count)

```
-------------------------------------------------------|
| 2 | * | 2 | 1 |   |   |   | 2 | * | 4 | * | * | 4 | 2 |
-------------------------------------------------------|
| * | 4 | * | 2 | 1 |   |   | 2 | * | 4 | * | * | * | * |
-------------------------------------------------------|
| 2 | * | 3 | * | 2 | 1 |   | 1 | 1 | 2 | 2 | 4 | * | 4 |
-------------------------------------------------------|
| 1 | 1 | 2 | 2 | * | 2 | 1 | 2 | 1 | 1 |   | 1 | 2 | * |
-------------------------------------------------------|
|   |   |   | 1 | 2 | 3 | * | 2 | * | 1 |   |   | 1 | 1 |
-------------------------------------------------------|
|   |   |   |   |   | 1 | * | 2 | 2 | 1 | 1 |   |   |   |   |
-------------------------------------------------------|
|   |   |   |   |   | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 |   |
-------------------------------------------------------|
| 1 | 1 |   |   |   | 1 | * | 3 | * | * | 3 | * | 2 |   |   |
-------------------------------------------------------|
| * | 1 |   |   |   |   | 1 | 3 | * | 4 | 3 | 5 | * | 3 |   |
-------------------------------------------------------|
| 1 | 1 |   |   |   |   | 2 | * | 2 | 1 | * | * | 2 |   |   |
-------------------------------------------------------|
|   |   |   |   |   |   | 1 | 2 | 2 | 2 | 2 | 2 | 1 |   |   |
-------------------------------------------------------|
|   |   |   |   |   |   |   | 1 | * | 1 |   |   |   |   |   |
-------------------------------------------------------|
```

Pseudocode:

```
// minesweeper.cpp
// Midterm Exam, SCI120-145, Spr 2025
//
// NOTES:
//  1. DO NOT DEVIATE from the pseudo-code algorithm below. Use identifiers and types
//     mentioned and do not use keywords that we haven't covered in class.
//  2. User defined identifiers are shown surrounded by < >. DO NOT TYPE THE < > as part
//     of the names.
//  3. Arrays index may be shown as subscripts, e.g., A_k,  which is the k-th element of
//     the array A
//  4. Floating-point types can be any floating point types you find appropriate. Also,
//     integer types can be any integer type you find appropriate.
//  5. Most math functions are shown in math or plain text format. You must convert them
//     into C++ syntax.
//  6. See exercise instructions for i/o and user interface examples
//
// PROGRAM STARTS BELOW THIS LINE ----------------------------------------------------
```

```
// CODE: Include only all the necessary headers and namespaces


// --- MineSweeper STRUCTURE -----------------------------------------------------
//  CODE: Define a structure called <MineSweeper>. The structure contains the
//  following members:
//      1. CODE: Define two integer-type variables called <rows> and <columns>
//      2. CODE: Define a character-type variable called <msymb>
//      3. CODE: Define a string-type variable called <board>


// --- addCount FUNCTION ---------------------------------------------------------
//  CODE: Define a function called <addCount> that returns  a <MineSweeper>-type value.
//  It takes three parameters: two integers named <row> and <col> and a
//  <MineSweeper>-type named <game>. The body contains the following:
//      1. CODE: Write a loop with integer counter <r> for r = row-1, row, row+1
//      The loop contains the following:
//          - CODE: If <r> is greater or equal than zero and <r> is less than <rows> of
//          <game>, do the following:
//                A. CODE: Write a loop with int counter <c> for c = col-1, col, col+ 1
//                The loop contains the following:
//                  * CODE: If <c> is greater or equal than zero and <c> is less than
//                  <columns> of <game>, do the following:
//                      1) CODE: Define an integer-type variable called <k>, initialized
//                      to <r> times <columns> of <game> plus <c>
//                      2) CODE: Add 1 to the [k]-th element of <board> of <game> if the
//                      <k>-th element of <board> of <game> is not equal to <msymb> of
//                      <game>. (Note: don't use the increment operator or the combined
//                      operator to add one)
//      2. CODE: After the loops, return <game>
// NOTE: in the function above, there is an if statement inside a loop, which is inside
// an if statement inside a loop



// --- addMinesNCount FUNCTION ---------------------------------------------------
//  CODE: Define a function called <addMinesNCount> that returns a <MineSweeper>-type
//  value. It takes two parameters: a <MineSweeper>-type named <game> and an integer-type
//  named <mines>. The body contains the following:
//      1. CODE: Define an object of <random_device>-type called <rd>
//      2. CODE: Define an object of <uniform_int_distribution>-type for unsigned
//      integers named <pick>. The distribution should be from 0 to the number of
//      elements in <board> of <game> minus 1 (i.e., its last index)
//      3. CODE: Write a loop which executes as long as <mines> is not zero. The loop
//      contains the following:
//          A. CODE: Define an integer-type named <i> initialize with a random value
//          generated by <pick>
//          B. CODE: Define an integer-type named <j> initialize with a random value
//          generated by <pick>
//          C. CODE: set <i> to <j> if the <j>-th element of <board> of <game> is grater
//          than the <i>-th element of <board> of <game>
//          D. CODE: if the <i>-th element of <board> of <game> is a character numeric
//          digit (i.e., single digit in character form. You can use an STL function for
//          this), then perform the following statements:
//              1) CODE: Set the <i>-th element of <board> of <game> to <msymb> of <game>
//              2) CODE: Call <addCount> with the following arguments:
//                  <i> divided by <columns> of <game>,
//                  <i> modulus <columns> of <game> (remainder), and
//                  <game>
```

```
//              Save the returned value back in <game>
//              3) CODE: subtract one from <mines>
//      4. CODE: after the loop, return <game>


// --- displayBoard FUNCTION -----------------------------------------------------------
//  CODE: Define a function called <displayBoard> that does not return a value. The
//  function takes one parameter: a <MineSweeper>-type named <game>. The body contains
//  the following:
//      1. CODE: Define an object of string-type called <trail>, initialized to "|\n"
//      2. CODE: Define an object of string-type called <line>, initialized with a
//      consecutive dashes, '-', with a length equal to <columns> of <game> times four.
//      3. CODE: Display to the standard output <line> followed by <trail> (don't add an
//      additional new line).
//      4. CODE: Write a loop with integer-type counter <r>, for all rows of <game>,
//      i.e., starting from 0 to <rows> of <game>. In the loop:
//              A. CODE: Write a loop with integer-type counter <c>, for all columns of
//              <game>, i.e., starting from 0 to <columns> of <game>. In the loop:
//                      1) CODE: Define an integer-type variable called <k>, initialized
//                      to <r> times <columns> of <game> plus <c>
//                      2) CODE: Define a character named <s>, initialized with the
//                      <k>-th element of <board> of <game>.
//                      3) CODE: Set <s> to ' ' only if <s> is '0'
//                      4) CODE: Display to the standard output "| ", followed by <s>,
//                      followed by single space (with no additional new line).
//              B. CODE: After the <c> loop, display to the standard output <trail>,
//              followed by <line>, followed by <trail> (with no additional new line).


// --- main FUNCTION -------------------------------------------------------------------
//  CODE: The body of <main> contains the following:
//      1. CODE: Define a <MineSweeper>-type structure variable named <game>. Use a
//      initializer list to initialize its first three member to 12, 14, and '*',
//      respectively.
//      2. CODE: resize the number of elements of <board> of <game> to <rows> of <game>
//      times <columns> of <game>. Set all those elements to '0'. (Hint: all this can
//      be done with a single statement).
//
//      3. CODE: Call <addMinesNCount> with arguments <game> and 30, and assign its
//      return value back to <game>
//      4. CODE: call <displayBoard> with argument <game>


//  END OF CODE
```