CS/SE 1337 – Extra Credit Program #3

The Chaos Game: Surprising Structures from Randomness

Dr. Doug DeGroot's C++ Classes

Due: Friday, April 26, 2019, by midnight

How to submit: Upload your entire project – the source code and all screenshots as described below – in a zip file to the eLearning site. The project must compile on either Codeblocks or MS Visual Studio 2015 (or later). Include your name in the file/project name, like so:

EC3-CS1337-*nnn*-John-Doe

where *nnn* is your section number (either 003 or 010).

Maximum Number of Extra Credit Points: 40 points added to your total homework and test scores. To earn the maximum number of extra credit points, you must implement the steps described in Steps 1 through 9 below. For *extra* extra credit, implement Step 10 (five points) and several aspects of Step 11; document in a separate file what experiments you performed and show snapshots of your results.

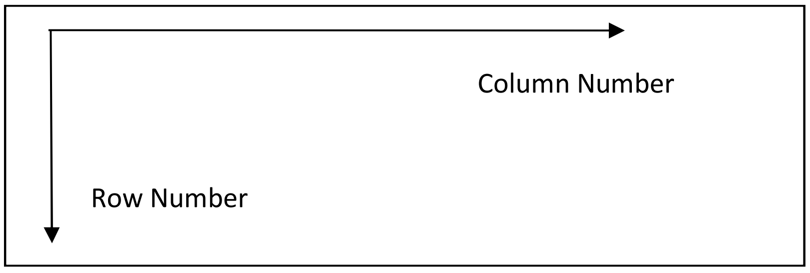
In your submission, you must add some brief comments in the top of your program stating what steps you implemented and what, if any, experiments you performed so that the grader and I know what to look for in your code.

Discussion:

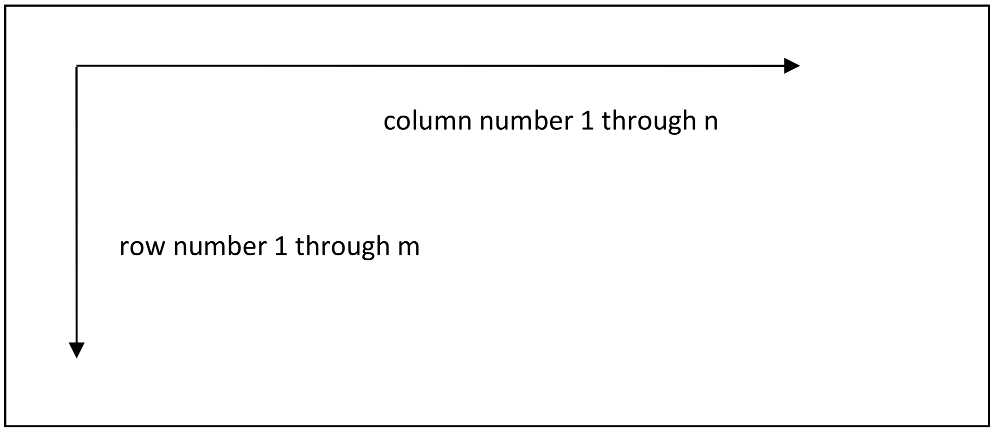
For this extra-credit programming assignment, you will have only part of the information about the program's intended result. Hopefully, what you see as the final result will be both surprising and mysterious. But above all, it should be interesting to watch. (Actually, we've already seen some of the resulting outputs during class; so you know what to expect.)

Step 1:

Consider the coordinate system of the computer's console; it is numbered by Row and Column (just as in Excel). It is not the same as the x-y coordinate system of the Cartesian plane. It is typically visualized as follows:



The cells are numbered by row-column coordinates, with row 0 being at the top and column 0 on the left. We are going to map a Boolean grid (2D array) onto the console just as we've done before. As stated, the coordinate system will be enumerated in the row-column format, and so will the coordinates of every point you draw on the console. To print (draw) the grid on the console, you simply print the top row (row 1), from column 1 to column n, then the second row, from column 1 to column n, and so on. This will give you a screen layout as shown below.



In this coordinate system, the upper left corner of the console is at location (1,1) and the lower right corner is at location (m,n).

Use a 2D array of bools of with dimensions (size) at least 50x60. Let's call those two dimensions something like maxRows x maxCols.

Step 2:

Now we want to draw various points in the grid and display them on the screen. Define three points in your program that will be located somewhere in the grid. Use a struct to define the abstract data type *point*, something like this:

```
        struct point {
                int x;
                int y;
        }
```

To make your three points, you can define variables p1, p2, and p3, or you can define an array of 3 points or anything else – that's your call.

Step 3:

For the first experiment, use your three points to define a triangle that sits in the grid. You could define your three points (vertices) of the triangle with coordinates

```
        x = 0, y = maxCols/2
        x = maxRows, y = 0
        x = maxRows, y = maxCols
```

Step 4:

Create a function that accepts a point as a parameter and extracts the x and y coordinates from the point and then sets the cell at the x-y location within your Boolean grid to true.

Step 5:

Create one new point, say curPoint, at some random location with the grid; it doesn't matter where you create this first point. Turn this point on, too.

Step 6:

Now iterate the following process some large number of times, say 1000.

Pick one of the three points at random. Call it something like *chosenPoint*.

Calculate the midpoint between *curPoint* and *chosenPoint*, and set *curPoint* to that midpoint. (You can easily calculate the x and y coordinates of this midpoint as shown in class.)

Turn that cell within the grid to on.

Draw the grid on the console.

Then repeat this process the specified number of times. (To speed up the simulation, you can set a parameter to redraw the grid every nth iteration – just like skip-ahead simulation in the Game of Life.

Step 7:

To draw the grid, I suggest printing "X " for the turned-on cells (note the extra space character) and two spaces (" ") for the cells that are off. But of course, you can experiment with your own printing choices.

Step 8:

When the program finally stops iterating, take a screenshot of the resulting drawing that appears on your console and submit it with your homework. Call it "3node-drawing" or something similar.

Step 9:

Now go back and change the number of initial points from 3 to 4. Set the coordinates of these four points to the corners of the grid (making a rectangle). Then do as before: create a new point called curPoint (or whatever) anywhere within the grid, turn it on, select one of the four corners at random, calculate the midpoint of the chosen point and the current point, etc., etc., just as before. Iterate some large number of steps and take a screenshot of the final drawing; call it something like "four-points". Submit this drawing with your homework, too.

Step 10 - optional:

Finally, do this same process for 5 starting points. Try to set the 5 "corner" points to those representative of a pentagon. See what final drawing results. Take a screenshot of this too and submit it with your homework.

Step 11 - optional:

To have even more of a challenge, try some experiments with the code such as

1. Not allowing your program to choose the same randomly selected node twice in succession. See whether this makes any significant difference in your final images. (hint: It definitely should.)
2. Instead of moving to a point midway between your current point and the randomly selected point, try moving some other fraction of the distance to the chosen node, say 75% or 33% instead of 50%.
3. Experiment, experiment, experiment! Save any "impressive" diagrams and submit them too along with a short explanation in your code explaining what you did and how you derived this diagram.

Annotations for the code:

1. The main function can be at either the beginning or the end of the program. I don’t care which.
2. Add comments at the top of your main.cpp file to include your name, the name of the program, and notes on how your design works when executed. Point out any special features or techniques you added using a comment saying “// Special Features.”
3. Comment your code effectively, as we discussed in class. Use descriptive variable names everywhere so that the code becomes as self-documenting as possible. Use additional commentary to improve readability and comprehensibility by other people.
4. You absolutely MUST use consistent indentation and coding styles throughout the program. Failure to do so will result in a loss of three points.
5. No late submissions will be accepted since this the end of the semester is quickly coming.. Please meet the deadline. And please don’t ask for an extension, as grades will have to be turned in shortly after the end of the semester.
6. Please be sure to include as many diagrams (screenshots) as you think are illustrative of the (interesting) experiments you run.
7. In your submission, you must add some comments in the top of your program stating what steps you implemented and what, if any, experiments you performed.