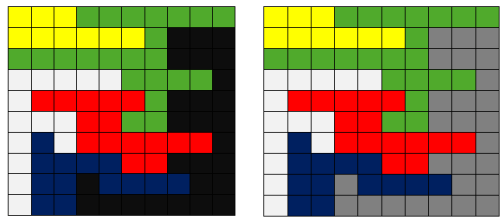
Project: Flood Fill

Prerequisite: Chapter 13

If you’re familiar with Photoshop or other image editing applications, you have probably used flood fill tool. Flood fill is used to transform all adjacent pixels of the same color into a new color.

At its simplest, you tell the flood fill 2 things: the pixel where it should start, and a new color. Flood fill then looks at the pixel where you tell it to start, sees the current color of that pixel, then changes that pixel *and all pixels touching the starting pixel which were the same color* to the new color. Then the process propagates outward and the same rule is applied to all pixels adjacent to the starting pixel which had their color changed.

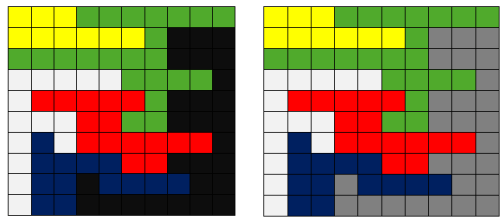
For instance, here is a very simple image. Each square represents a single pixel:



If we imagine this image as an array with [0][0] in the upper left corner, then we could flood fill starting at [1][7] and replace it with the color gray. Flood fill then

* Looks at [1][7] and notes that it contains black. It changes [1][7] to gray.
* Looks at all the pixels around [1][7] including diagonally for pixels which are the original color of [1][7], black. It finds [1][8], [2][8], and [2][7].
* It turns [1][8], [2][8], and [2][7] gray.
* For each pixel which it turned gray in the last step, it looks at all the pixels around it for black pixels and turns them gray.
* The process continues until the entire black region connected with [1][7] has been turned gray.

Once this process is completed the image has become:



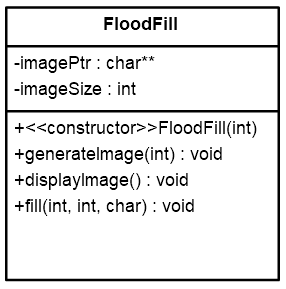
For this project, you’re going to implement a class that does a flood fill.

# Requirements

At a minimum, you will create the classes OrderedPair and FloodFill.

OrderedPair will be a simple class that can store 2 integers. A single OrderedPair object will store a set of values representing the row and column of a pixel in your image. Use good object-oriented design techniques for OrderedPair.

FloodFill will be more complex:



You may add additional private variables and functions as needed but your public methods *must* match the above (your code will be tested with an instructor-provided main()).

Here are some details for the members of FloodFill:

|  |
| --- |
| imagePtr  A pointer to a 2-dimensional char array on the heap. Each char in this 2-dimensional array represents a pixel in an image. |
| imageSize  The size of the array currently pointed to by imagePtr. The array will always be a square (same number of rows and columns). |
| FloodFill(int)  Constructor which initializes the object to contain a randomly generated array of the given int size. For example, if int is 10, then the array is 10x10. |
| generateImage(int)  Creates a new array pointed to by imagePtr of size int. For example, if int is 10, then the array is 10x10. The array should be randomly populated with the chars ‘c’, ‘m’, ‘y’, and ‘k’ with each char being equally likely. The char stored in each element represents a color. |
| displayImage()  Displays the current chars in the array pointed to by imagePtr. Should also display row numbers above the image and column numbers to the left of the image. Row and column numbers should start over at zero once they pass 9. Also, all char should have a single space between them. For example:  0 1 2 3 4 5 6 7 8 9 0 1  0 y c y c y y c m y k m m  1 y y m c k c c m m m m m  2 y y m k c k k c m y k m  3 m c m k c y m y y k m m  4 k c k c y c c k m c k k  5 y y y k c y y m m m m c  6 k y m k c m y k y y c y  7 k m y m y k y c y m y k  8 y c y k m m c c k m c m  9 m c c k m m k k c y k k  0 c y c m m y k c c c m k  1 y c m m m m c c k c y c |
| fill(int row, int col, char newColor)  Flood fills beginning at [row][col] and replacing color there with newColor. fill() should display to the screen:  Beginning at [row][col] I will flood fill and replace {whatever color is currently at [row][col]} with newColor.  Implement the flood fill algorithm below using the [Standard Template Library queue](http://www.cplusplus.com/reference/queue/queue/). Note that the STL queue uses pop() for dequeue, push() for enqueue, and front() for peek.   * + Create an empty queue that can contain OrderedPair objects.   + matchColor = Original pixel color at indicated [row][col] where fill begins. matchColor is the color which will be replaced in the fill process.   + Enqueue the pixel at [row][col] and mark it processed (you’ll need some way to track which pixels in the image have been processed and which have not)   + While the queue is not empty     - currentPixel = peek the queue     - Pop the queue     - Set currentPixel’s color to newColor     - For all 8 pixels adjacent to currentPixel       * If pixel has *not* been processed and color is matchColor enqueue the pixel and mark it processed |

# Turn In

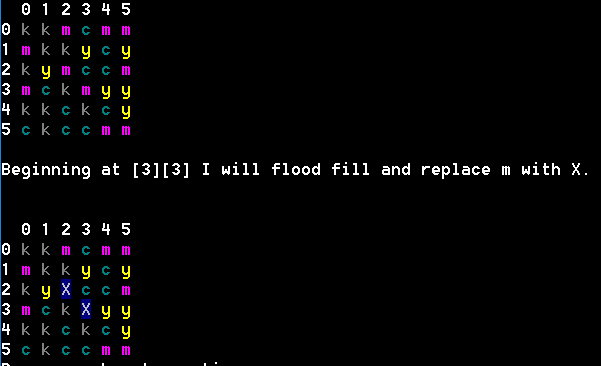
Upload FloodFill.h, FloodFill.cpp, OrderedPair.h, OrderedPair.cpp, and all other files you created to complete your project before the due date.

# Sample Output

Here are some colorized examples of program output. Color has been added to make the fill more obvious. You **do not** have to colorize your output (though you may if you like). It has been colorized [using the method here](https://www.daniweb.com/programming/software-development/code/216345/add-a-little-color-to-your-console-text) which only works on Windows. These samples display the initial randomly generated image, then call fill(), then display the image after the flood fill.

Jacob Kier

Johnson County Community College

Sample 1

Sample 2