CSc 127B — Introduction to Computer Science II
Fall 2015 (McCann)

http://www.cs.arizona.edu/classes/cs127b/fall15/

Program #7: Boundary-Fill

Due Date: October 29th, 2015, at 9:00 p.m. MST

Overview: If you've ever used a paint program (such as Microsoft Windows' Paint, or the multi-platform GIMP), you've probably used the 'fill' tool that lets you easily fill a bounded region with a color. One way to implement such a tool is with an algorithm known as Boundary–Fill. For this assignment, you'll implement a version that uses a queue to remember work that still needs to be done.

Boundary–Fill requires an image, a location within the image, and a new color to be used for filling. To keep things simple, we'll use a 2D array of characters as our image, meaning that locations will be (row,column) pairs and colors will be represented by different characters. Our goal is to replace all locations in the connected region defined by the initial location's character with the new character. Matching characters are part of the same region if one can be reached from the other by following a path that consists of only that same character and that can be traversed without following a diagonal direction (that is, without moving NE, SE, SW, or NW). For example, consider Figure 1, below. The '@' at (3,1) is not part of the region defined by the other '@' characters. The '+' at (3,2) is part of the region of '+' characters because it's 'north' of the '+' at (4,2). A change of the '+' at (4,2) to another character would create three separate connected regions of '+'.

```
   0123456789     0123456789     0123456789     0123456789
  +----------+   +----------+   +----------+   +----------+
 0|@@@@@@@@@=|  0|@@@@@@@@@=|  0|@@@@@@@@@=|  0|@@@@@@@@@=|
 1|**@@@**@@=|  1|**@@@**@@=|  1|**@@@**@@=|  1|**@@@**@@=|
 2|++@@@**@@=|  2|1+@@@**@@=|  2|..@@@**@@=|  2|..@@@**@@=|
 3|+@+@@@@@@=|  3|.@+@@@@@@=|  3|.@+@@@@@@=|  3|.@1@@@@@@=|
 4|++++++====|  4|2+++++====|  4|1+++++====|  4|......====|
 5|===+++====|  5|===+++====|  5|===+++====|  5|===2++====|
  +----------+   +----------+   +----------+   +----------+
   Figure 1       Figure 2       Figure 3       Figure 4

   ----------     ----------     ----------     ----------
   (3,0)          (2,0)(4,0)     (4,0)          (3,2)(5,3)   <--- Queues
   ----------     ----------     ----------     ----------
```

The version of Boundary–Fill that we'll use for this assignment uses a queue to keep track of the left–most endpoints of horizontal spans of characters that still need to be changed. Here's how it works. In Figure 1, let's start with location (3,0), which is a '+'. We add it to the queue, shown below the figure. The rest of the algorithm continues so long as there are locations held in the queue. After removing (3,0) from the queue, we change it and all of the '+'s directly connected to it on row 3 to the new character ('.'). Here, only (3,0) is changed. Next, we look for connected spans of '+' both directly above and directly below the span of '+' that we just changed. In this case, there's a span above and a span below. We enqueue the left–most location of those spans (in this case, (2,0) and (4,0), marked with '1' and '2' for illustration purposes only), as shown in Figure 2. In processing row 2, we find no new spans of '+' (Figure 3), and so no locations are added to the queue, leaving just (4.0) on the queue. Dequeuing and processing (4,0) reveals two new spans (Figure 4). You should be able to see how the algorithm will complete from there.

(Continued ...)

Assignment: Write a complete, well-documented Java program that implements the version of the Boundary–Fill algorithm described above. The image format is described in the **Data** section, below. We know that Java doesn't provide a dedicated queue class, so we'll write our own, with our own queue interface. You are to implement a queue interface named `CS127BQueueInterface` and a queue class named `CS127BQueue` that implements your interface, and represents queues using the circular array queue representation we just learned. (This means no JCF classes; you are restricted to ordinary arrays.) The exact method names, method arguments, etc., are up to you. The Boundary–Fill algorithm will be implemented as part of the `Prog7` class, and will, of course, make use of your queue class.

Write your main method in the `Prog7` class to accept the name of the file holding the image, the row and column coordinates of the starting point, and the character to be used to fill the area from the command line. For example:

```
java Prog7 pretty.dat 23 7 "*"
```

(The asterisk is in quotes because punctuation symbols on the command line can cause strange behaviors otherwise.) If the user doesn't give all four parts, prompt the user for the information:

```
Enter the name of the image file:  pretty.dat
Enter the row,col starting location of the fill area, comma-separated:  23,7
Enter the character to be used for filling:  *
```

Data: A sample 'image' is available from the class web page: `prog7sample.dat`. The first line has two integers, separated by at least one space. The first number is the number of rows in the image, the second the number of columns. The rest of the lines consist of characters, one image row per line.

As usual, create your own images for testing purposes. Feel free to share your testing images with classmates. The section leaders will create their own set of testing images to use when they grade your programs, and no, those won't be shared with you before the due date.

Output: For each image, location, and fill character specified, your program is to: (a) display the original image, location, character at that location, and the new character to be used for filling; (b) display the queue content after each pixel span has been filled and the left ends of all adjacent pixel spans have been queued (similar to the queue content given in the examples above); and (c) display the final image and the counts of both (i) the number of pixel spans filled and (ii) the total number of pixels that were modified. You need not display the row and column labels or the borders that appear in the samples, but you may if you wish to.

Turn In: When you're done with the assignment and are ready to submit it, turn in all of your files (`Prog7.java`, `CS127BQueueInterface`, and `CS127BQueue`) to the `cs127bsXp07` directory at any time before the due date and time. Of course, you can turn them in late if you still have late days to use, or, if your late days are exhausted, you don't mind losing 20% per day.

Want to Learn More?

- Filling regions with color (or a 'texture' of a pattern of colors) is a common graphics operation for which multiple algorithms exist. Any comprehensive computer graphics text will cover Boundary–Fill (sometimes under the name 'Flood–Fill').

Hints, Reminders, and Other Requirements:

- As the queue needs to hold locations, you may use Java's `Point` class to hold the coordinates. Thus, your queue would be represented by an array of `Point` objects. You may write your own location class, if you prefer.
- If you get ambitious, you can replace the text output with a graphical representation of the image. This could allow the user to watch the image being filled. No extra credit, of course, but it would be fun to do and would impress your SL.
- Finally, my usual admonition: **Start early!**