CSS 342 Programming Assignment #4 Due: Mar. 8 (design due Feb. 22) **Winter 2017**

Linked lists and recursion

In this program, we are going to use recursion to perform image segmentation and linked lists to store some results.

Due dates:
Design: **February 22** (bring two hard copies to class – see below)
Implementation: **March 8** (use the Catalyst dropbox)
Your design should consist of documented header files for any classes that you expect to use, documentation (comments and prototypes) for each function that you plan to use in your main program, and pseudocode (algorithms) for the important functions/methods (in all classes and other code). (This would include loops and non-trivial function calls, including recursion.) You will be able to change the design after you turn it in, but this will not improve the score you get on the design. On February 22, we will hold design reviews in small groups in class. **For full credit on your design, you must bring two hard copies of your design to class that day**.

Important

The default stack size in Visual C++ 2015 is not sufficient to run recursive algorithms that can perform thousands of recursive calls. In order to increase the stack size in your workspace, perform the following steps:

1. In Solution Explorer, **select your project**.
2. In the Project menu, **select Properties** (in Configuration Properties).
3. In the Property page that pops up, click to open **Linker** and then click on **System**.
4. In the Stack Reserve Size box type: 100000000 (7 zeros, you might even need 8)

Definitions

Neighbor pixel: Each pixel has four neighbors (the pixels just above it, below it, to the left of it, and to the right of it) unless it is at the boundary of the image, in which case it has two or three neighbors.
Connected group: A set of pixels in the image in which each pixel is "connected" to each other pixel. For pixels *a* and *b* to be connected they must be in the same group. In addition, they must either be neighbors or *a* must be connected to a neighbor of *b* through other pixels in the group.
Image segmentation: A division of the image into disjoint, connected groups of pixels that share similar attributes. The attribute that we will be using is the color of the pixel.

Assignment

One way to segment the image into connected groups of pixels is to start each group with a seed pixel that does not currently belong to any other connected group. The seed pixel is the start of a new group and each of the neighboring pixels is then examined (recursively). For any neighbor that is not already in a group, if the color of the neighbor is close enough to the seed pixel, then that pixel is added to the group and each of its neighbors are examined recursively. The recursive process continues until every pixel that has a "close enough" color to the seed pixel and is connected by another such pixel to the seed pixel (and is not in another group already) is added to the group.

To decide whether a pixel has a color that is close enough to the seed pixel we will use the following test. Let *seed* be the seed pixel and *p* be another pixel, then we will say that the color is close enough if:

abs(seed.red – p.red) + abs(seed.green – p.green) + abs(seed.blue – p.blue) < 100

In order for me to test your code, the procedure must be repeatable. If your code is correct, you should get the same result as my implementation. For this to be the case, we need to examine the possible seed pixels in a pre-determined order. The order that I want you to use is called the raster order in the image. The first row is examined in left-to-right order, then the second row in left-to-right order, then the third row, until all of the rows have been examined. Note that only pixels that have not already been added to a group should be used as seed pixels.

Your output image should consist of a new image similar to the input image, except that the color of each pixel in the output is the average of the colors of each pixel in the group of the corresponding pixel in the input image (round down). *You should also output*

some information to the console (see below).

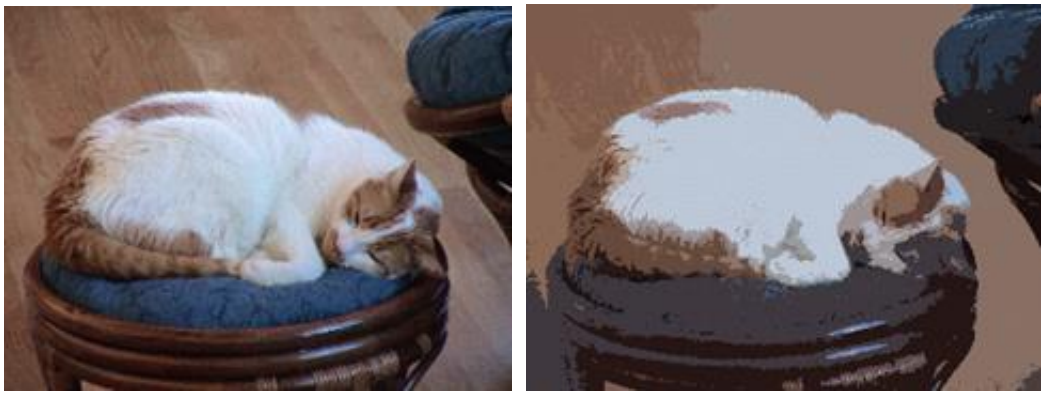
In order to accomplish this, I want you to write a container class for pixels (don't make it a template class) that uses a linked list to store a set of image pixels (and as much additional information as you like, such as pixel location). The container class should have at least the following methods: constructor, copy constructor, operator=, destructor, addPixel, and merge (see below.) You can implement other member functions (including size and averageColor), if desired.

The merge operation should take a single parameter, similar to the copy constructor, but it should result in "this" being a container with both sets of pixels in it. The other input container should not be changed. To test your merge operation, merge each group into a single container after you are done processing it. At the end of your program, this should be a container with a linked list of all of the pixels in the image (not a linked list of containers). Make sure to test your operator= and copy constructor, even if you don't need them in the driver. I will.

Finally, output to the console the total number of segments found (don't include the big merged group), the number of pixels in the merged group, and the average color of this merged group.

For testing your program, you can use one of the previous test images or here is a new one:
http://courses.washington.edu/css342/cfolson/program4/test.gif

For one particular test image, the results of my implementation can be seen below.



Segmentation results on an example image

Note that the segmentation operation should not be implemented as methods in your image class or the container class. This implementation should be part of main() or functions called from main(). The container class and your image class should be loosely coupled (neither needs to know about the other). As in the previous assignments, your main program should read in a test image called "test.gif" and output the result to "output.gif."

Suggestions

The only objects that you need (in addition to the input image) are an image in which to store results and two objects to store your connected groups of pixels as you are processing them (the current one and the merged one). You may assume that none of the pixels in the input image is completely black (0, 0, 0). So, you can use this color as a marker to indicate pixels that have or haven't been processed in one of your images.

Overall, you should have an outer loop over the pixels in the image that selects seed pixels. For each pixel you find that is not part of a group, yet, make it a new seed pixel. Call the recursive function to find all of the pixels in the connected group. You can then determine the average color and color the appropriate pixels in the output image (and merge the new group into the container that contains all assigned pixels).

To determine if a pixel has already been assigned to a group, check the color of the pixel in the output image (black = no, any other color = yes). To prevent infinite recursion, you will also need to mark the pixels in the current group in the output image. You can set

the pixel to be any color (other than black), since it will be overwritten with the average group color later. So, your recursion will probably look something like:

- Check base case(s).
- Add new pixel to group (must both modify the pixel container and mark the pixel in the output image).
- Check pixels to the left, right, above, and below using recursive calls.

Grading

This program is worth 40% of the programming score for the course (15% for the design, 25% for the implementation). See the grading rubric for a breakdown on how each program is scored.