CS 117: Introduction to Computer Science	**Winter 2006**
Handout 7: Problem Set #4 (Due: 16 and 19 February 2006)	*10 February 2006*

> *PICTURE, n.*
> *A representation in two dimensions of something wearisome in three.*
> — Ambrose Bierce (1842–1913/1914?), *The Devil's Dictionary.*

In this assignment, your task is to write some new methods for the PhotoLab class—giving you additional practice with loops and arrays, and with bigger classes, more complicated methods, and so forth.

- **Part I** of this assignment is a group assignment, and is due on Thursday, 16 February 2006, 11:59pm via hsp. Put your work in a directory called `ps4I`.
- **Part II** of this assignment is an individual assignment, and is due on Sunday, 19 February 2006, 11:59pm via hsp. When your group work is done, copy your joint solution into your own account into a directory called `ps4II`.

For images, you can again use the images from the last assignment (background1.jpg, background2.jpg, background3.jpg, amy.jpg, or dave.jpg), or obtain your own image instead if you like: you can use your picture from the Carleton directory or you can take a picture with a digital camera.

You can start again from scratch, or you can use the code that you wrote for the last assignment as a starting point. The PhotoLab that you submit this time around will have a different (additional) set of methods, so it doesn't matter if you had problems last time with some of your methods. A couple of the methods here I suggested that you work on in lab a few weeks ago, so if you did you'll have a head start.

Here are the methods that your PhotoLab class should have:

- `public PhotoLab(EzImage image)`

  The constructor; the input parameter is the source image.

- `public EzImage rotateCounterclockwise()`

  Creates and returns a new image that is rotated 90 degrees counterclockwise from the original.

- `public EzImage rotateClockwise()`

  Creates and returns a new image that is rotated 90 degrees clockwise from the original.

- `public EzImage reflectHorizontal()`

  Creates and returns a new image that is a horizontal mirror-image of the original.

- `public EzImage reflectVertical()`

  Creates and returns a new image that is a vertical mirror-image of the original.

- `public EzImage crop(int rowStart, int rowStop, int colStart, int colStop)`

  Creates and returns a new image that is the result of cropping your image. Extracts only those pixels from pixels rowStart through rowStop and colStart through colStop (all inclusive). If any of the parameters are out of range for your image, you should just return the original image.

- `public EzImage circleCrop(int centerRow, int centerCol, int radius)`

  Creates and returns a new image resulting from cropping the image to a circular-shaped portion of the original with a center at `centerRow` and `centerCol` and a radius of `radius`. Because the image itself must still be rectangular to fit inside a window, make the background of the rest of the image white. If centerRow, centerCol, radius, or some combination thereof are invalid, just return the original image.

- `public EzImage dither()`

  Dithering is a technique used when you want to print a gray picture in a place such as a newspaper where no shades are available. Instead, you need to use individual pixels of black and white to simulate shades of gray. A standard algorithm for this task is the *Floyd–Steinberg algorithm*, which works as follows:

  1. Loop over all pixels, running across one row at a time, then each pixel within that row as you move from left to right.
  2. For each pixel that you encounter: if its value is larger than 128, set it to 255 (pure white). Otherwise, set it to 0 (pure black). Record the "error", which is the old value of this pixel minus the new one.
  3. This "error" represents how much blackness we have added to this pixel. Distribute this blackness to adjacent pixels as follows:
     - Add 7/16 of the error to the pixel immediately to the right ("east").
     - Add 3/16 of the error to the pixel immediately diagonally to the bottom left ("southwest").
     - Add 5/16 of the error to the pixel immediately below ("south").
     - Add 1/16 of the error to the pixel immediately diagonally to the bottom right ("southeast").

     Be careful near the edges of the image! If some "error" should be distributed that doesn't exist (e.g., it's off the edge of the image), you should just ignore it.

  Tips:

  - Dithering should only be done on grayscale images—actually you can dither color images, too, but it's more complicated—so use the EzImage instance method `copyToGrayScale()` to convert an image to gray before you get started.
  - *In order for dithering to work, you must make your changes to the same image that you are looping over.* Dithering by reading one image and making your changes on a copy will not work correctly, because the "error" never gets a chance to accumulate. In other words, make sure that you make a gray scale copy of the image first, and then do all dithering work (looping, reading, writing, etc.) on the copy itself.

- Also create a class `PhotoLabTester` to test your program. You're encourage to amuse yourself with images other than the ones that are provided.

  Thus concludes Part I of this assignment. It's to be turned in via hsp by 11:59p on Thursday, 16 February 2006.

Part II: to be done individually.

Add the following additional method to your PhotoLab class:

- `public EzImage blur(int radius)`

  Creates and returns a blurry version of the image. For each pixel, average all of the pixels within a circle centered at that pixel and with a radius as given. Make sure to put all of your results in a new array, instead of overwriting your original as you go; otherwise, your blurred pixels will cascade on top of each other. Keep in mind that some approaches to programming this result in much slower algorithms. On a machine roughly comparable to the lab machines, I can blur background3.jpg with a radius of 10 in a few seconds. Any technique that works will receive most of the points for this problem, but you need to end up with timing close to these numbers if you wish to earn it all.

Submit Part II of this assignment via hsp by 11:59p on Sunday, 19 February 2006.