The graphics purpose of this project is to implement and explore the simplest line detection algorithm (of Prewitt) and to compare the outcomes when it applied to the same image before and after blurring. Line detection is important modern ways to make computers "understand" images, say for robotic car driving.

The computer programming purpose is to gain further skill in
- coding loops, calculations and conditionals,
- solving problems about locating variables or objects (for us, Pixe}s) LOCATED by x, y coordinates in a Picture. The problems include dealing with nearby Pixels and with rectangular subregions (without getting array indexing out-of-bounds errors!)
- adapting sample codes from a textbook to a more complex situation by composing algorithms. Composing means run one after the other.
- and coding your own static methods in your own application class to create new Pictures. In past projects, you just added to a textbook class an instance method that modifies this Picture or World.

Readings:
- Looping through (a) all Pixels and (b) a rectangular subregion of a Picture is covered throughout the textbook and course so far.
- Section 5.2.1 covers copying all or parts of a Picture.
- Section 6.5 covers blurring.
- Section 6.2.5 covers the primitive edge detecting algorithm (a simplification of an algorithm published by Prewitt).
- To learn to work solidly with this and many other applications, study, in ALL of Chapters 5 and 6, ALL the applications of loops, locating objects like Pixels with x, y locations, copying data like color intensities, parametrized methods (both making and using them), blending with weighted average calculation, Pythagorean theorem for both geometric and color distance, and if statement conditionals applied to aspects of color and location. To become a REAL EXPERT: Copy, test and MODIFY some of them to do new things you make up yourself!

Testability requirement: It must be testable with the P3Tester program (on the Web): Otherwise it will be rejected with a 0 and to get credit (with 20 points deducted automatically), you must give your TA a testable revision. For full credit, it must reproduce our results on the butterfly1smaller image and work ok on some surprise images.
Code and TEST a Java class named `P3` with method

```java
public static Picture makeP3Picture(Picture source)
```
This method must make, copy color data into and return a `Picture` constructed by

```java
new Picture(source.getWidth() * 3, source.getHeight() * 2)
```
The returned picture has space for 6 pictures the same size and shape as the `source`. These 6 areas are to be filled in as follows:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A exact copy of Picture</td>
<td>Each pixel has gray scale value equal to the absolute value of</td>
<td>The image after edge detection, computed from the gray scale</td>
</tr>
<tr>
<td>source</td>
<td>the differences of averages computed from the source Picture</td>
<td>image just to the left. See textbook sec. 6.2.4 for the</td>
</tr>
<tr>
<td></td>
<td>as in Program 37, page 184.</td>
<td>explanation.</td>
</tr>
<tr>
<td>A blurred copy of Picture</td>
<td>Same as above except it is computed from the blurred copy of</td>
<td>Same as above except it is calculated from the gray scale</td>
</tr>
<tr>
<td>source</td>
<td>the source Picture instead of the original</td>
<td>image from it's left, which was made from the blurred copy.</td>
</tr>
<tr>
<td>blurred as explained in section 6.5 of the textbook.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Some questions and answers

1. Is there extra credit? Yes! To get it, you must first make an on-time submission that conforms to our testability requirement (REALLY!). Then, working with 3 images, butterfly1.jpg and two others, at least one you get yourself (NOT from G&E, but take your own or get one from the Web), try to do the best edge detection you can with variations of "numPixels" for blurring and "amount" for detection threshold, possibly other algorithms. Write a word-processed report with a 3-5 page written description of what you did and the results, and include an informative selection of the 3 original images and examples of bad, better, and best results for each, and maybe others. Acceptable reports will earn up to 50 points just added to your project score (like another 1/2 project!) Email reports to the Prof. within a week of the project due date.

2. Lateness? No extra credit, and the usual 14% off per 24 hours late.

3. What (a) blurring "numPixels" value and (b) edge detecting "amount" did you use to get the sample output, and what values should we use?

I used 2 for numPixels and 10 for amount; use those in your submission.

4. How should I structure my program?
I can think of two good ways: Either is OK. WAY One is to write 3 methods: A method to return a new blurred Picture (made from a Picture parameter), a method to return the new line detection Picture (made likewise), and a third to copy a smaller Picture into a specified place of a bigger Picture.
WAY Two: Write a method to copy the original into a bigger Picture, plus two others: one to blur and one to detect lines. The blur and line detect methods take as parameters the result Picture and two rectangular regions of it. Both methods access the colors from the first region and change the colors in the second region.

5. How should I begin?
First, get your program to copy the source Picture into the upper-left 6th of the target Picture. MAKE SURE IT COMPILES AND RUNS PROPERLY! NOW!

Second (or third), write and test code to copy the source WITHOUT CHANGE AGAIN into the lower-left 6th. When that works, (NOT BEFORE) modify that code to do the blurring. (You must put in if statements now to avoid crashes!)

Third (or second), write and test code to copy the source without change AGAIN into the upper-middle 6th. When that works, (NOT BEFORE) modify that code to make the gray scale image. (Tricky: NOT ALL gray scale Pixels will be changed!) After the third step, you'll have no trouble finishing!

6. How do I make the differences of the averages (on textbook page 184) into a gray-scale color? Easy: Use a variable like `diff` to hold the absolute value that G&E's code computed, and use

```
new Color((int)diff, (int) diff, (int) diff)
//RGB intensities all the same: that's gray!
```