

IC210 Fall 2007

Programming Project 2

Image Manipulation

Note: this project relies on material covered in classes “Arrays & Pointers I” to “Arrays and Pointers IV”. If you wish, you may read ahead if you want to get a head start on the project. Note though that you can do the basic menu without any of this array material.

Pre-Coding Analysis: Due by “close of business” on Thurs Oct 18, 2007. Turn in:

1. A printout of the image that you will use for testing (converted to PGM format), and
2. Flowchart for Function 2 (Load Image), AND
3. Flowchart for Function 5 (Create the image negative)

Flowcharts may be either neatly hand-drawn or using the RAPTOR tool. Don't worry at this point details on how the array is passed to your function (or returned), but on what the function needs to do to create/manipulate the array. Make sure that your flowchart clearly shows how you intend to solve the *conceptually difficult parts* of the project. Note that you DO NOT have to actually code up the solution as part of the Pre-Coding Analysis.

1. Note that this Pre-Coding Analysis is NOT a routine out-of-class assignment, but rather is part of the project and therefore subject to the Department's policy concerning programming projects.
2. Recall that the Department's policy states that for a programming project **“midshipmen may give no assistance whatsoever to any person and may receive no assistance whatsoever** from any person other than the midshipman's instructor for the course assigning the project.” Please view <http://www.cs.usna.edu/academics/ProgrammingPolicy.pdf> for additional details on this issue.

Executive Summary

Images on a computer are not stored in the same way that images on film are stored. A photographic image is a recording of the visible light reflected from the objects in the camera's field of view which is then saved to film through a chemical reaction between the light and the film. A digital camera also stores the light reflected from objects within its field of view. The difference is the manner in which the image is stored and displayed. A digital image, whether taken by a digital camera or scanned from a film reproduction, is displayed as a series of pixels (**picture element**), individual points capable of displaying one color or light intensity at a time. This information is then stored in a computer's memory as an array of numerical values, representing each pixel's color.

For this project, you will write a program that is capable of loading an image, performing various manipulations on the image, collecting important image statistics, and saving the image to disk. All of these features will be implemented through a menu system with the various tasks performed by functions that your `main()` function will call.

Due Dates and Honor

The Pre-Coding Analysis will be due by the “close of business” on **Thurs Oct 18**. The project will be due by the “close of business” on **Monday October 29, 2007**. See the course policy for details about due dates and late penalties.

Again, this is a *Programming Project*, and it is very important that you understand and abide by the Department's policy concerning programming projects. Please view: <http://www.cs.usna.edu/academics/ProgrammingPolicy.pdf>

Extra Information

For this project you will be loading and manipulating a grayscale .pgm (Portable Gray Map) image. The Portable Gray Map image format is the lowest common denominator grayscale file format. It is designed to be extremely easy to learn and write programs for. Consider the following test image:

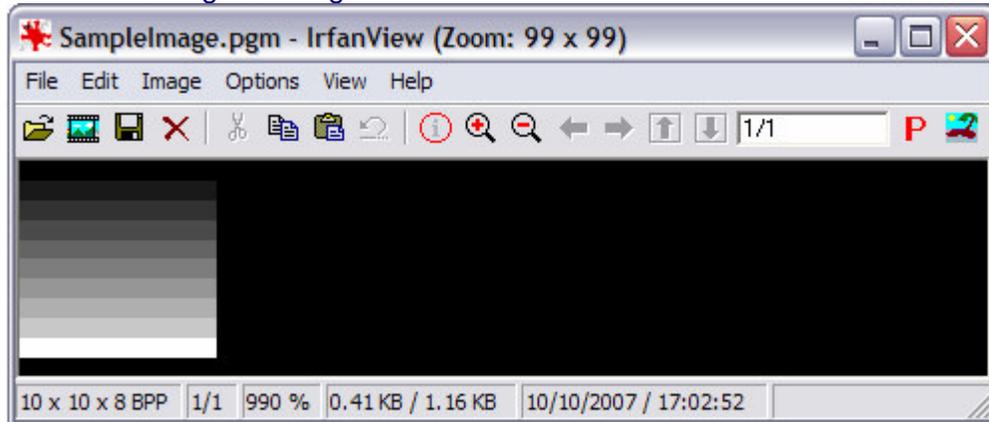


Figure 1. A 10x10 Pixel Image Shown with an Image Viewing Application

This image is described by a text file that looks like this if viewed by WordPad:

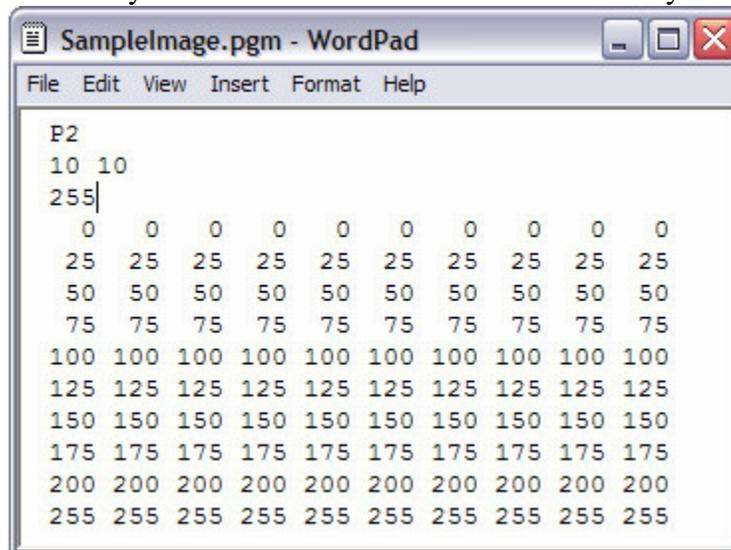


Figure 2. The Same 10x10 Image Viewed Using WordPad

The simplicity of the .pgm format is due to how the image information is stored. Like the .wav file from Project 1, the .pgm image file consists of two parts: a simple header and the individual pixel values. The header and data fields are saved in a file in their ASCII form which makes reading, manipulating, and saving the file relatively easy.

The file format definition is shown below. Compare this with the file shown in Figure 1:

- The header chunk contains the following pieces of information:
 1. A “magic number” which identifies the file type. A .pgm’s magic number is “P2” (yeah, we know it’s not really a number, but that’s what it’s called).
 2. Whitespace (blanks, TABs, CRs, LFs).
 3. The image’s width, in pixels, formatted in ASCII (in this case, 10).
 4. Whitespace.
 5. The image’s height, in pixels, formatted in ASCII (in this case, 10).

6. Whitespace.
 7. The maximum gray value (Maxval), formatted in ASCII. Must be less than 65536 and greater than zero. Usually, Maxval is 255. For this project assume your Maxval is 255.
 8. A single whitespace character, usually a newline.
- The data chunk consists of (*width x height*) pixel values separated by whitespace. The values will range between 0 (black) and the Maxval (white), inclusive. The first value represents the upper left most pixel of the image (in this case a 0). Each subsequent value represents a pixel continuing from left to right in the first row, then continuing with the next row. The final value represents the bottom right most pixel (in this case, 255). Although it is not required by the format definition, it is common practice for a newline character to be inserted after every *width* pixels (i.e., the pixels for each row in the image appear to be on a separate line of the file).

Take time now to look at Figure 2 and make sure you understand why it produces an image that looks like Figure 1!

Our PGM format only handles different shades of gray. More complex formats handle color, usually by providing more than one number for each pixel, to represent the amount of red, green, and blue in each pixel (RGB).

Details

The project is divided up into several functions, worth varying number of points, **not strictly based on difficulty**. You are strongly encouraged to solve each function - including thorough testing. Your maximum grade will depend upon which functions you get working.

All of your program's functionality will be implemented via function calls, with your `main()` function serving primarily as central hub with some sort of selection structure calling the functions that implement the required functionality. The user should be able to continue selecting options until he chooses to quit.

You will be able to use any **appropriate** picture you wish. You can use the `i_view32.exe` program included with this project to convert a `.jpg`, `.gif`, or most other image formats into a `.pgm` file. You can also use `i_view32` to view the results of your program.

To convert an image to a `.pgm` for use in your program follow these instructions:

1. Start `i_view32` and open the file you wish to convert.
2. Select "File/Save As" and from the "Save as type" drop down menu select "PGM – Portable Graymap".
3. A popup window will appear to the right of the Dialog box. The middle section of the popup will say "PBM/PGM/PPM" with two options. Ensure the "Ascii encoding" radio button is selected.
4. Type the name you want to name your file and select "Save".
5. Once the file is created, open it with WordPad, **not Notepad**, and delete the line that says, "# Created by IrfanView".
6. Save the file and you are done.

Program Functionality—



Figure 3 An Example Image

Function 0: (10 pts.) Display menu.

Using a function, print a menu to the screen and return the user's menu selection to the `main()` function. The menu should include a selection option for all the features your program is capable of; as well as a quit option.

```
"X:\C210\Projects\Project 2\Debug\main.exe"
Welcome to LT Johnson's SHIP, <S.H. Image Program>

Please choose an option below:
1) Load Image
2) Save Image
3) Image Properties
4) Blur Image
5) Create Negative
6) Flip Image along Vertical Axis
7) Flip Image along Horizontal Axis
8) Insert Message
9) Extract Message
10) Quit
>> _
```

Figure 4 A Sample Menu. Options are not shown in the order that you will do them.

Function 1: (20 pts.) Load Image.

This function should:

- Ask the user for the filename to be opened
- Open the file. If this fails then print an error message and return to main.
- Read the header
- Create an appropriately sized array to store the image data
- Load the image data into the array
- Return the data array, image width, and image height back to `main()`; it's your job to figure out an appropriate way to do this.

Function 2: (15 pts.) Save Image.

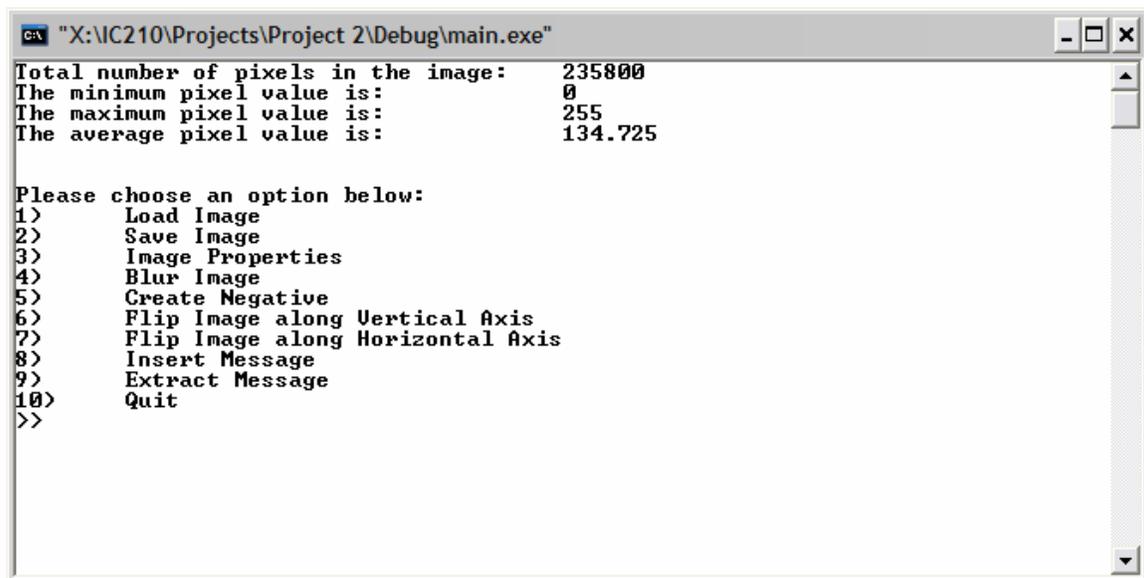
Save your image in a properly formatted .pgm file. The function should be passed all the required data and then prompt the user for the name of the file he wishes to save the image in. The save file should be saved in the format described in the **Extra Information** section.

1. For readability's sake each line of pixel values should be saved on a separate line of the file as shown in Figure 2.
2. Columns of pixels should be right justified with each other (as in Figure 2).

Be sure to close the file, using `.close()` command, once your program is done writing to the file and before returning to the `main()` function.

Function 3: (15 pts.) Display Image Properties.

This function will display to the screen the total number of pixels, the minimum and the maximum pixel value. Additionally, you will calculate and display the average pixel value.



```
CA "X:\C210\Projects\Project 2\Debug\main.exe"
Total number of pixels in the image:    235800
The minimum pixel value is:           0
The maximum pixel value is:           255
The average pixel value is:            134.725

Please choose an option below:
1)    Load Image
2)    Save Image
3)    Image Properties
4)    Blur Image
5)    Create Negative
6)    Flip Image along Vertical Axis
7)    Flip Image along Horizontal Axis
8)    Insert Message
9)    Extract Message
10)   Quit
>>
```

Figure 5 Image statistics for example.pgm

Function 4: (15 pts.) Create the Image Negative.

Write a function that creates the image negative. In other words black become white and vice versa. For example the negative of the following pixel values:

0 185 166 133 83 76 90 255

is:

255 70 89 122 172 179 165 0

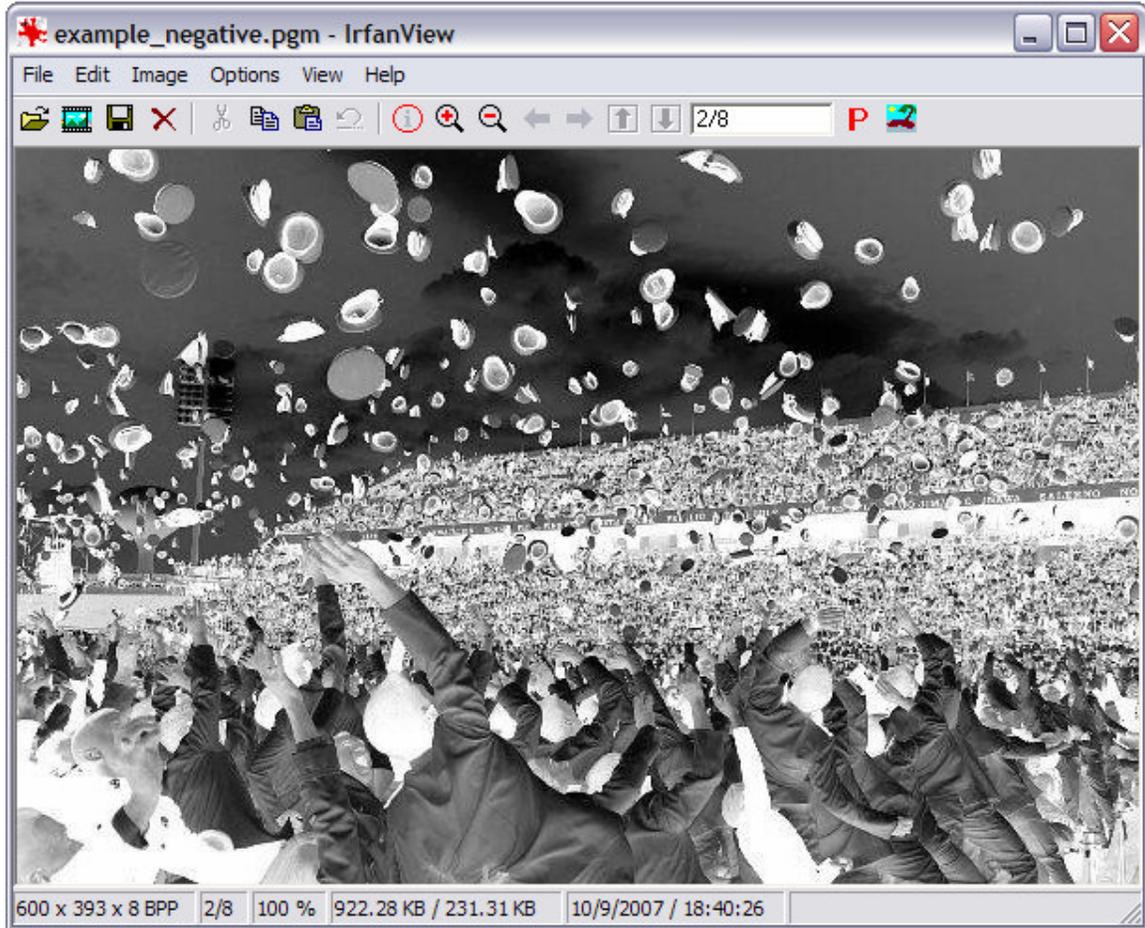


Figure 6 The negative of example.pgm

Note: the remaining functions are somewhat more challenging. **Start early!!**

Function 5: (10 pts.) Flip the Image along the Vertical Axis.

Write a function which flips the image along a vertical line through the middle of the image. In other words the left side of the image will become the right and the vice versa.



Figure 7 example.pgm flipped along the vertical axis

Hint: Get Function 5 to work first and then apply what you've learned to get Function 6 to work.

Function 6: (10 pts.) Flip the Image along the Horizontal Axis.

Write a function which flips the image along a horizontal line through the middle of the image. In other words the top of the image will become the bottom and the bottom will become the top.

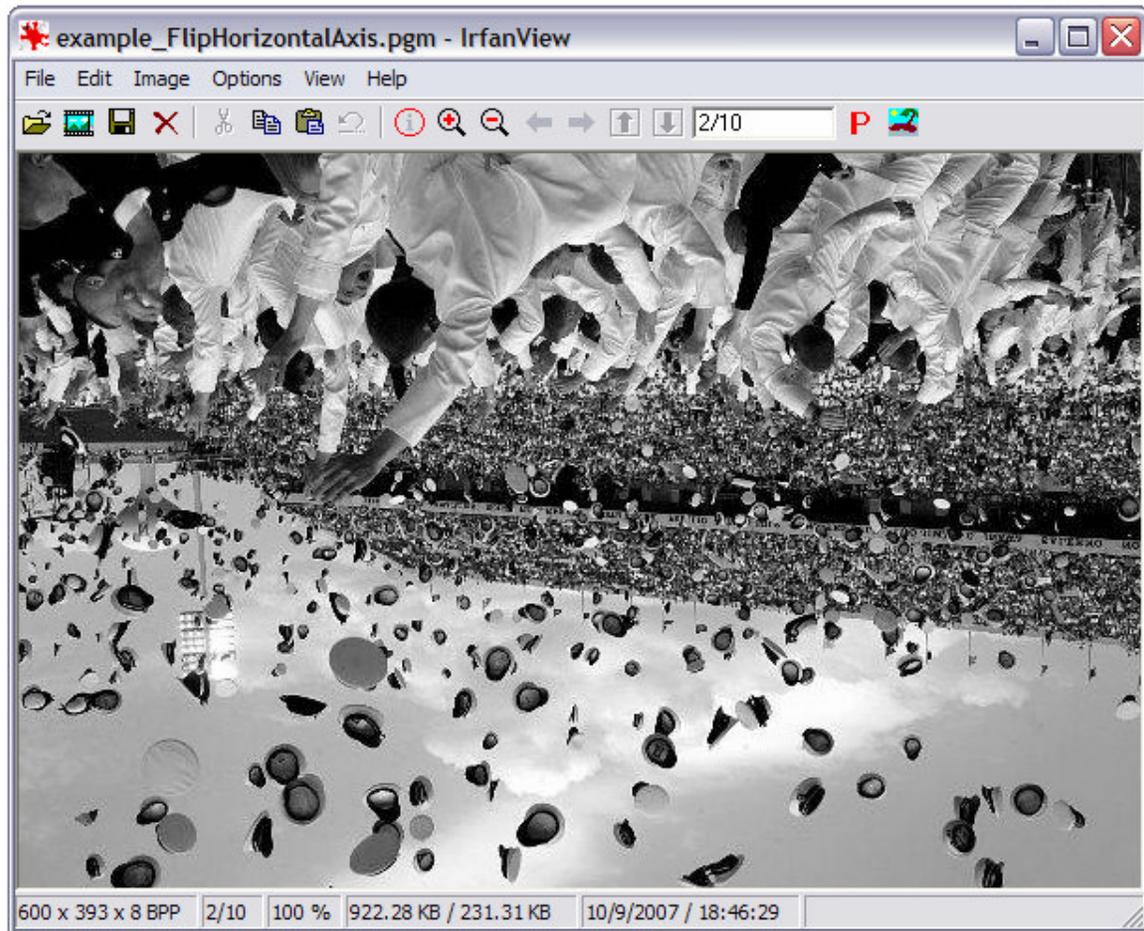


Figure 8 example.pgm flipped along the horizontal axis

Function 7: (Extra Credit – 4 pts.) Insert a Secret Message within the Image.

Steganography is the art and science of writing hidden messages in such a way that no one apart from the intended recipient knows of the existence of the message. A very common form of steganography is to hide a message in an image by changing the **least significant bit** of eight pixels to represent a single bit of an ASCII character.

Conceptually speaking, if you wanted to hide the character 'M' (ASCII = 77, binary = 01001101) in an image with the following pixel values:

221 221 220 218 218 190 189 173

You would encode it by transforming each pixel value into its binary representation and then substituting the corresponding bit from the message into the least significant bit position. Note that the bits that are changed are underlined.

221 = 11011101 →	0 =	11011100	= 220
221 = 11011101 →	1 =	11011101	= 221
220 = 11011100 →	0 =	11011100	= 220
218 = 11011010 →	0 =	11011100	= 218
218 = 11011010 →	1 =	11011011	= 219
190 = 10111110 →	1 =	10111111	= 191
189 = 10111101 →	0 =	10111100	= 188
173 = 10101101 →	1 =	10101101	= 173

Write a function that asks the user for a text file. Assume the file being encoded is relatively small and you do not need to check to ensure the picture is large enough. Your function will then encode the message in the image. Encode an ETX (end of text) character (ASCII value = 3) to signal the end of your message.



Figure 9 There's a hidden message in here. Can you see it?

Hint: Use the `.get()` function to ensure that you encode the whitespace properly.

Function 8: (Extra Credit – 3 pts.) Retrieve a Secret Message from the Image.

The `.pgm` image included as a sample has a message hidden within it. The message is encoded in the manner described in **Function 7**. Write a function that is capable of retrieving messages hidden in an image and saving it in a text file defined by the user. Do not output the ETX character to the output file.



Function 9: (Extra Credit – 3 pts.) Blur the Image.

In this blurring scheme each pixel will be replaced with the average value of the pixel itself, as well as the pixel that came before it and the pixel that comes next in the array. For example if the array goes from 0 to N the i^{th} pixel will be replaced with the value, $(A[i-1] + A[i] + A[i+1])/3$. For the end cases weight the 0^{th} and N^{th} double. In other words, the 0^{th} pixel is blurred by with the value, $(A[0] + A[0] + A[1])/3$, and the N^{th} pixel is blurred with the value, $(A[N-1] + A[N] + A[N])/3$.



Figure 10 example.pgm after quantization

Hint: Be sure to use the original pixel values when quantizing.

What to submit and how it will be graded

You will submit a **single** solution to that is capable of executing all of the functions that you implemented. You will indicate on your coversheet which functions you have implemented and wish to be graded.

Note: If a function you submit for grading does not work as it should the grading penalties will be substantial. Make sure that any function you submit for grading works properly.

The pre-coding analysis is worth 5 points. The project is worth 100 points and the **maximum grade** you can earn is based on the summation of the pre-coding analysis and the points for each function that you submit and that **works correctly**.

The grading breakdown is as follows:

• Pre-coding Analysis	5 pts.
• 0. Display Menu	10 pts.
• 1. Load Image	20 pts.
• 2. Save Image	15 pts.
• 3. Display Image Properties	15 pts.
• 4. Create Image Negative	15 pts.
• 5. Flip Image Along the Vertical Axis	10 pts.
• <u>6. Flip Image Along the Horizontal Axis</u>	<u>10 pts.</u>
Total	100 pts.

Important grading points:

- Your program should work for any valid .pgm file (not just the one you test with)
- Appropriate use of functions to implement the program is critical to receiving the maximum possible score.
- **If your program does not compile as submitted**, you will receive a zero.
- If your program does not give correct results, penalties will be substantial. If you can't get a function to work properly, do not submit it for grading.
- Your program must read input and write output in the exact format specified in this project description.
- Your program's source code must comply with the Required Style Guide in order to maximize the grade you receive on this project.
- Your grade will also depend on the reasonable use of C++ code. Don't use 50 lines of code where 5 would do.

There will be both a paper and an electronic part to your submissions. The paper submission can be handed to your instructor in class, slid under their office door, or put in their mailbox. For the purposes of any late penalties, your project is not considered submitted until your instructor receives BOTH the electronic and paper portions of your submission.

Electronic submission: Unless otherwise specified by your instructor, your electronic submission should be a single email message with title "**IC210 Project 2 Submission**". Include in this email:

- your project2.cpp file
- the .pgm file you used to test your program

If you have any questions, send a separate message with a different subject.

In addition, you will submit a hard copy of your source code, a screen shot of your original image as well as a screen shot of the results for each implemented function your program is capable of. Attach the Project 2 cover sheet to your project ensuring that you note on the sheet which of the functions you implemented and want graded.

(continued on next page)

Paper submission: staple in this order:

- Cover sheet, with signed honor statement
- Printout of your code. Make sure the indentation is correct on the printout! See notes on printing on the course homepage
- Printout of your image, after conversion to PGM
- Screenshots for each function that works. Be sure to LABEL each one. Here's what to show for each step:
 - Functions 0, 1, 2 – no screenshot to submit
 - Function 3 – screenshot of console output when run on your PGM file
 - Function 4 – screenshot of IMAGE after reversed by your program
 - Function 5 – screenshot of IMAGE after flipped by your program
 - Function 6 – screenshot of IMAGE after flipped by your program
 - Function 7 – screenshot of console where user is providing message
 - Function 8 – printout of file showing decoded message from sample file
 - Function 9 – screenshot of IMAGE after blurred by your program

TIPS:

- Follow the Style Guide as you go! It's easier to indent and comment as you write than at the end, and you'll make it easier on yourself to write your program. And you'll maximize your grade by following the guidelines.
- Compile often so you don't end up with a snake's nest of errors all at once
- Save a backup copy of your program's source code in case your sponsor's dog eats your X drive.
- **Start early!** Allow time for you to get new ideas and to see the instructor for help.
- Remember that this project is not to be discussed with anyone besides your instructor! Don't defeat your learning AND jeopardize the honor of yourself or your friends by talking to them about it.