CSCD 240 Homework 3 and 4 (Project)
Simple Image Processing using C pointers to pointers, string operations, type casting, math library, command-line parsing, or file I/O

This is a comprehensive project, which is weighted twice as much as a regular homework. That is, the whole project accounts for 200 points totally. When calculating the weighted final score, this project is considered as two homeworks.

Submissions
Please submit all header files, source files, test images and a makefile, zip them up into a single zip file. You are also required to create a pdf file that shows that you tested all test cases with different command line arguments. You have to include this pdf file into your single zip file. Please name your zip file and your unzipped directory by your lastnameFirstinitial.zip.

Problems Description
In this project, we manipulate PGM images (Portable Gray Map). PGM files are pure ASCII text files, with header information and intensity value for each pixels in the image file. If you have an image ballon.pgm, you can use command 'less ballon.pgm' to explore its format. Surely you can install an image viewer to visualize the image with you naked eye. On windows, you can use **Irfanviw**, http://www.irfanview.com
On a Mac machine, you can download **ToyViwer** in your apple store for free.

The detailed format for PGM file can be found here, also you can download some sample PGM files, besides one included in this project package.
http://people.sc.fsu.edu/~jburkardt/data/pgma/pgma.html

The following is an example PGM file named smallFile.pgm

```
P2
# feep.ascii.pgm
24 7
15
0 0  0  0  0  0  0  0 0  0  0  0  0  0  0  0 0  0  0  0  0  0  0
0 3  3  3  3  0  0  7  7 7  7  0  0 11 11 11 11 0  0 15 15 15 15  0
0 3  0  0  0  0  0  7  0 0  0  0  0 11  0  0  0 0  0 15  0  0 15  0
0 3  3  3  0  0  0  7  7 7  0  0  0 11 11 11  0 0  0 15 15 15 15  0
0 3  0  0  0  0  0  7  0 0  0  0  0 11  0  0  0 0  0 15  0  0  0  0
0 3  0  0  0  0  0  7  7 7  7  0  0 11 11 11 11 0  0 15  0  0  0  0
0 0  0  0  0  0  0  0  0 0  0  0  0  0  0  0  0 0  0  0  0  0  0  0
```

The image header consists of the first 4 lines.
1. The first line 'P2', is a magic number to tell the image viewer that the file is a ASCII pgm file.
2. The second line starts with #, is a line of comment.

3. The numbers in the third line, means this image has 24 COLUMNS, and 7 ROWs of pixels in it.
4. The fourth line means the maximal intensive value in the image is 15. Each pixel has intensive value. **The bigger intensive value is, the brighter or whiter at that location is in the image. The intensive value 0 means a total black in the image.**

Intensive values for all pixels are listed after the file header, starting with line 5. These intensive values could be considered as a 2D array, with the row index and column index identifying each point or pixel at a particular location.

What you should do?

1) Implement the function that reads in the image as a text file using file I/O redirection. If you like to use the file I/O functions to read in the image, you get a 15 bonus points on top of 200 points.
2) Implement the function that paints a black dot (circle) in the image, you have to parameterize the center point and the radius of the circle you will draw.
3) Implement the function that paints a black edge frame in the image, you have to parameterize the width of the edge you will paint.
4) Notice that after you paint the edge or the black dot, you have to update the header to reflect the new maximal intensive value in the header.
5) Implement the function that writes back to a new image file that contains your painting. If you like to use the file I/O functions to write back to the new image, you get a bonus another 15 points on top of 200 points.
6) You are required to define the functions that are provided in the *.h file, you can NOT change the input, output and return type of these functions. You just need to define it in its corresponding *.c file. The purpose of this *.h file is to help you understand how pointers are used in real world applications.
7) You have to write code to parse the command-line arguments. When you run you program, you pass in the CircleCenter, Radius or Edge Width as command line arguments. For example,
   If the number of command line argument is not expected, your program are required to show a message:
   Usage:
   -e edgeWidth < oldImageFile > newImageFile
   -c circleCenterRow circleCenterCol radius < oldImageFile > newImageFile

   You have to run your program using command line should exactly look like:
   ./programName –e edgeWidth < originalImage > newImageFile
   to paint an edge of width of ***edgeWidth*** in the image of ***originalImage***

   ./programName –c circleCenterRow circleCenterCol radius < originalImage > newImageFile
   to paint an big round dot on the image with center at (***circleCenterRow, circleCenterCol***) and radius of ***radius***.

When user inputs wrong number of argument in command line or wrong input format, your program should not produce error, instead showing the Usage message above.

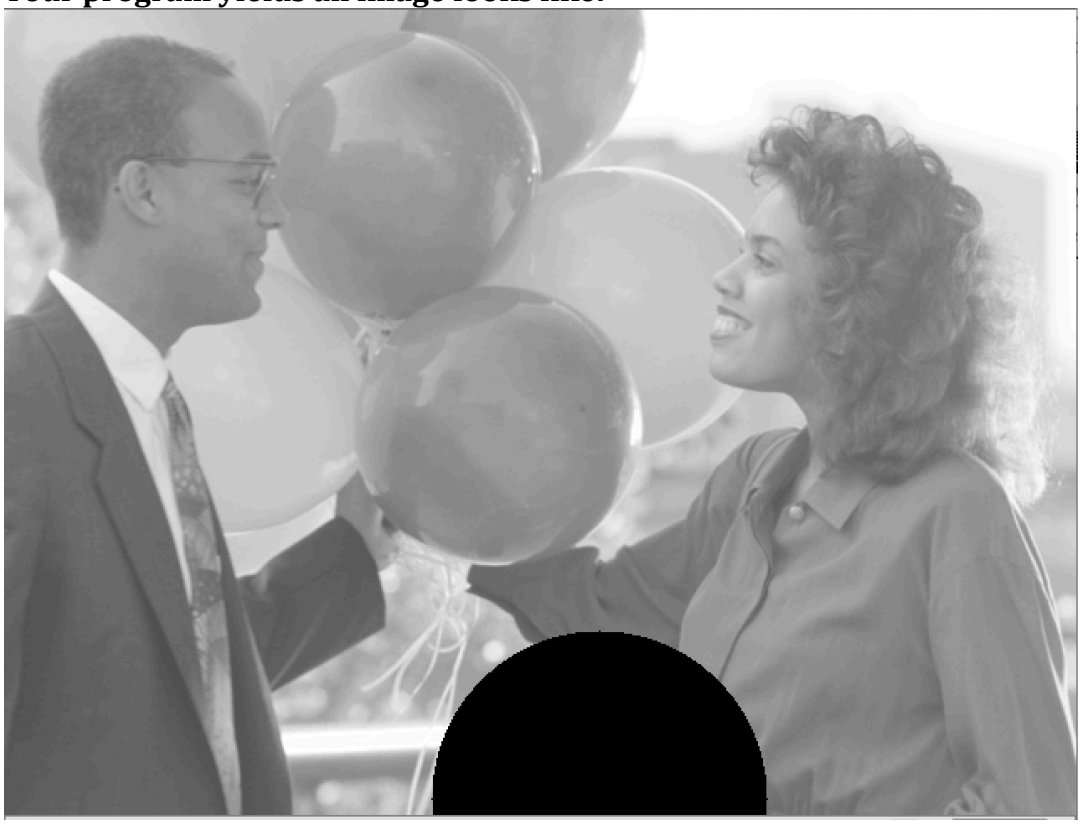
If your program could handle drawing a circle and an edge in your command line argument, you get another 20 bones points on top of 200.
In this case, you command line should look like the following, otherwise you can not get the bonus.
./programName -ce circleCenterRow circleCenterCol radius edgeWidth < originalImage > newImageFile

OR

./programName -c -e circleCenterRow circleCenterCol radius edgeWidth < originalImage > newImageFile

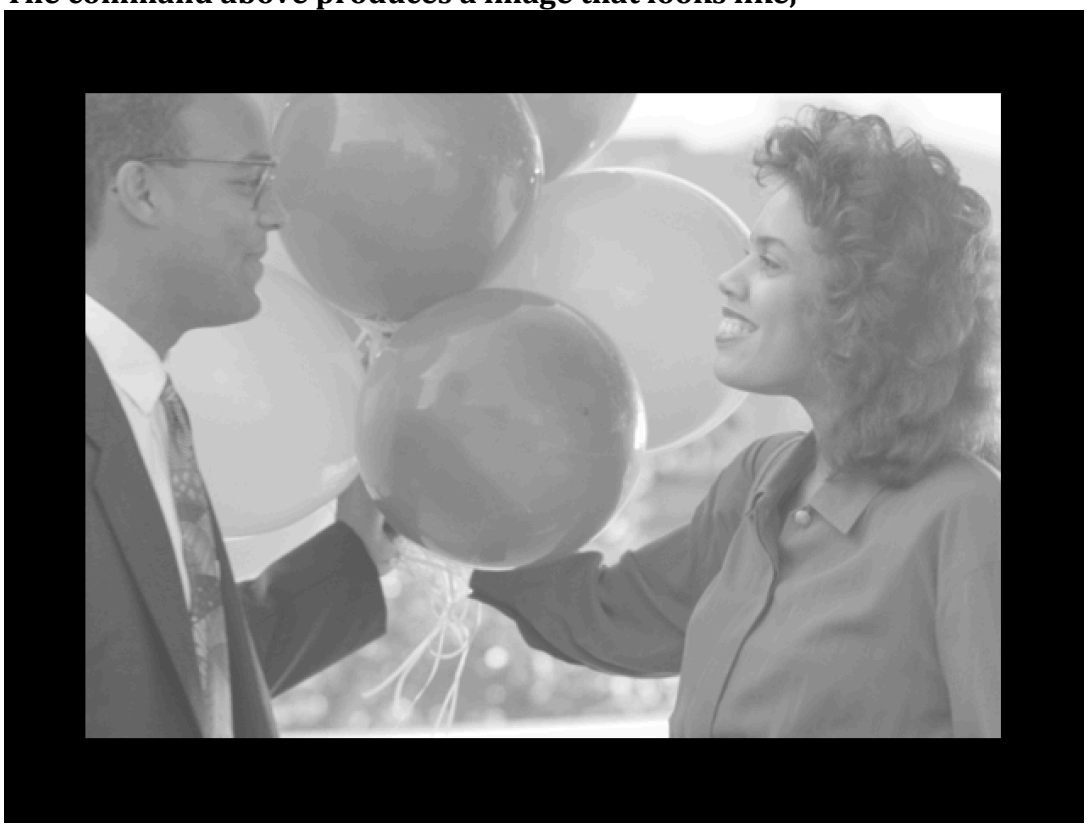
Test cases with the provided image

./myPaint -c 470 355 100 < ./balloons.ascii.pgm > balloons_c100_4.pgm
Your program yields an image looks like:



./myPaint -c 228 285 75 < ./balloons.ascii.pgm > balloons_c75_5.pgm
The command above yields an image,

./a.out -e 50 < ./balloons.ascii.pgm > balloons_e50_2.pgm
The command above produces a image that looks like,



If input wrong parameters:

./myPaint -e 50 300 < ./balloons.ascii.pgm
Usage:
-e edgeWidth < oldImageFile > newImageFile
-c circleCenterRow circleCenterCol radius < oldImageFile > newImageFile

./myPaint -e ab < ./balloons.ascii.pgm
Usage:
-e edgeWidth < oldImageFile > newImageFile
-c circleCenterRow circleCenterCol radius < oldImageFile > newImageFile

./myPaint -e < ./balloons.ascii.pgm
Usage:
-e edgeWidth < oldImageFile > newImageFile
-c circleCenterRow circleCenterCol radius < oldImageFile > newImageFile

./myPaint -c 470 355 < ./balloons.ascii.pgm
Usage:

-e edgeWidth < oldImageFile > newImageFile
-c circleCenterRow circleCenterCol radius < oldImageFile > newImageFile

./myPaint -c 470 355 50 60 < ./balloons.ascii.pgm
Usage:
-e edgeWidth < oldImageFile > newImageFile
-c circleCenterRow circleCenterCol radius < oldImageFile > newImageFile

./myPaint -c 470 90bc < ./balloons.ascii.pgm
Usage:
-e edgeWidth < oldImageFile > newImageFile
-c circleCenterRow circleCenterCol radius < oldImageFile > newImageFile

The original image before your processing looks like: