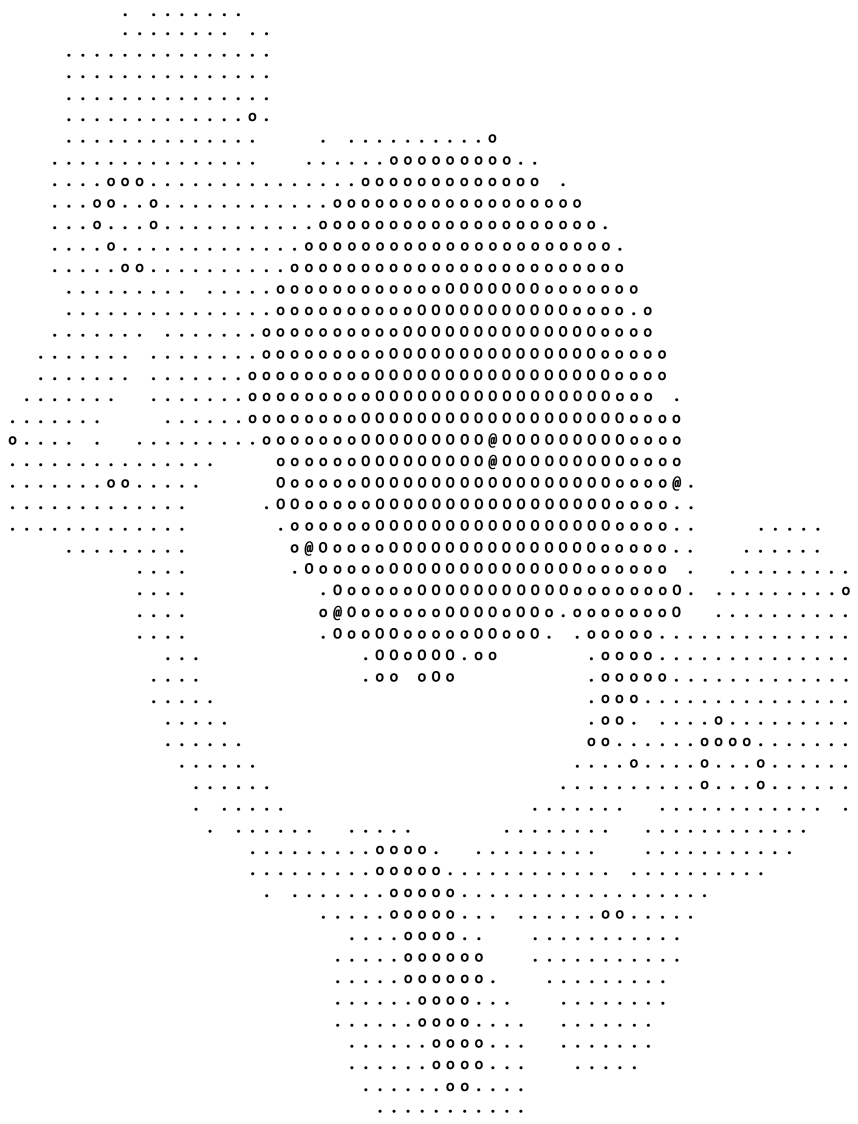
CSCI 135 Programming Exam #0
Fundamentals of Computer Science I
Fall 2014

This part of the exam is like a mini-programming assignment. You will create a program, compile it, and debug it as necessary. This part of the exam is open book and open web. You may use code from the course web site or from your past assignments. When you are done, submit all your Java source files to the Moodle exam #0 dropbox. Please ***double check you have submitted all the required files***. Before leaving the exam, you must ***checkout with the instructor*** to ensure your files are properly submitted.

You will have 100 minutes. No communication with anyone aside from the instructor is allowed. This includes all forms of real-world and electronic communication.

Grading. Your program will be graded on correctness and to a lesser degree on clarity (including comments) and efficiency. Partial credit is possible, so strive to provide a solution that demonstrates you know how to do as many parts of the problem as possible (even if there are bug(s) tripping up the total solution).



Overview. You are sick of all these flashy programs with high-resolution, true-color, 60 frames-per-second graphical shenanigans. You decide to go "old school" and write a program that converts a color image into ASCII art. In particular, you plan to use a simplified ASCII art scheme consisting of only the five characters: `. o O @` plus the space character.

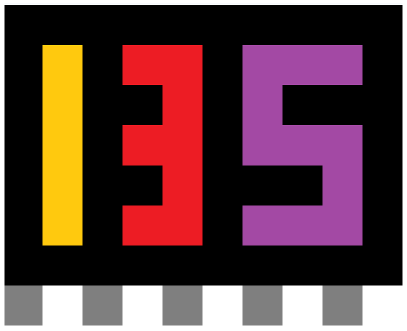
Your goal is to develop two programs: `Greyscale.java` and `Art.java`. The first program `Greyscale.java` converts a file containing image pixel data stored as RGB (red, green, blue) integer triples into floating-point greyscale values. It also computes the minimum, maximum, and average values of the image's greyscale values.

The second program `Art.java` take as input the output produced by `Greyscale.java`. It draws the ASCII art representation of the image. It can also optionally invert the greyscale colors and optionally convert to black and white only ASCII art based on a user-specified threshold.

A zip file containing stub programs, various test files, reference output, and `StdIn.java` can be downloaded from: art.zip

Part 1. `Greysacle.java` reads from ***standard input*** a special text image file format (the *.rgb files in the provided zip). These special files always start with two integers specifying the image's width and height in pixels. This is followed by a line for each row of pixels starting from the ***top*** of the image. In each row, there is a RGB (red, green, blue) triple for each pixel. Each value in an RGB triple is an integer in [0, 255]. So a row with 10 pixels would have a total of 30 integers (3 integers for each pixel).

Here is a 10 pixel wide by 8 pixel high sample image incredibly enlarged and its corresponding 135.rgb file:



```
10 8
  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   0   0
  0   0   0  255 201  14    0   0   0  237  28  36  237  28  36    0   0   0  163  73 164  163  73 164  163  73 164    0   0   0
  0   0   0  255 201  14    0   0   0    0   0   0  237  28  36    0   0   0  163  73 164    0   0   0    0   0   0    0   0   0
  0   0   0  255 201  14    0   0   0  237  28  36  237  28  36    0   0   0  163  73 164  163  73 164  163  73 164    0   0   0
  0   0   0  255 201  14    0   0   0    0   0   0  237  28  36    0   0   0    0   0   0    0   0   0  163  73 164    0   0   0
  0   0   0  255 201  14    0   0   0  237  28  36  237  28  36    0   0   0  163  73 164  163  73 164  163  73 164    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   0   0    0   0   0
127 127 127  255 255 255  127 127 127  255 255 255  127 127 127  255 255 255  127 127 127  255 255 255  127 127 127  255 255 255
```

As you can see, the black pixels are (0, 0, 0) triples, the white pixels are (255, 255, 255) triples, and other colors are somewhere in between. Each integer value in the file is separated from the next by one or more whitespace characters. You can assume the width and height will always be positive integers, the file will contain the correct number of triples, and each color value will be an integer in [0, 255].

The output of `Greyscale.java` starts with the width and height of the image in pixels (the same width and height from the *.rgb file). This is followed by a line for each row of pixels. Instead of RGB triples, you need to convert each triple into a single floating-point greyscale value. The greyscale value is a number in [0.0, 1.0] representing how light or dark a pixel is (0.0 = black, 1.0 = white). To convert an RGB color into a greyscale value use this formula:

$$grey = 0.299 \times red + 0.587 \times green + 0.114 \times blue$$

Note in the above equation, red, green, and blue refer to the proportion of a color on a scale from 0.0 (none of that color), to 1.0 (lots of that color). Before using the formula, you will need to convert your integer RGB values which are integers in [0, 255] to be a floating-point value in [0.0, 1.0].

After the pixel data, output three additional lines in the following order:

1. the minimum greyscale value of any pixel
2. the maximum greyscale value of any pixel
3. the average greyscale value of all the pixels

Here is a sample run:

```
% java Greyscale < 135.rgb
10 8
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.767953 0.000000 0.358443 0.358443 0.000000 0.432486 0.432486 0.432486 0.000000
0.000000 0.767953 0.000000 0.000000 0.358443 0.000000 0.432486 0.000000 0.000000 0.000000
0.000000 0.767953 0.000000 0.358443 0.358443 0.000000 0.432486 0.432486 0.432486 0.000000
0.000000 0.767953 0.000000 0.000000 0.358443 0.000000 0.000000 0.000000 0.432486 0.000000
0.000000 0.767953 0.000000 0.358443 0.358443 0.000000 0.432486 0.432486 0.432486 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.498039 1.000000 0.498039 1.000000 0.498039 1.000000 0.498039 1.000000 0.498039 1.000000
0.000000
1.000000
0.236936
```

Note in the above output each greyscale value has been rounded to 6 decimal places. Your program does ***NOT*** need to do this (though you can if you know how, but do not waste any time on this). Also note if you want to store your greyscale output as a file for use in part 2, you can redirect the output to a file:

```
% java Greyscale < 135.rgb > 135.grey
```

Part 2a. `Art.java` reads in from ***standard input*** a data file output from Greyscale.java. `Art.java` will first read in the image width (the number of pixels in each row) and image height (the number of rows of pixels). It will then read in the greyscale values of all the pixels, outputting text as it goes along.

Each row of pixels in the image should appear as a single line of text output. `Art.java` can safely ignore the final three minimum, maximum, and average values in the output format of `Greyscale.java`. Greyscale values are converted to ASCII art according to the following rules:

- If greyscale is in [0.0, 0.2), output as space
- If greyscale is in [0.2, 0.4), output as `.`
- If greyscale is in [0.4, 0.6), output as o
- If greyscale is in [0.6, 0.8), output as O
- If greyscale is in [0.8, 1.0], output as @

We have provided sample output files from our `Greyscale.java` solution. Thus you can implement `Art.java` even if your `Greyscale.java` program is not working properly. Here are several example runs:

```
% java Art < 135.grey

 O .. ooo
 O  . o
 O .. ooo
 O  .   o
 O .. ooo

o@o@o@o@o@

% java Art < check16.grey
           ....o
          ..oOoo
          .oOOOo
         ..OOOOo
         .OOOOoo
        .oOOOOoo
 ...   ..OOOOoo
.oo..  .OOO@Oo
.OOOo..o@O@Ooo
.OOOOooO@O@oo
.oOOOOO@O@Oo
 .oOOO@@@Oo
  .oOOO@@oo
   oo@@@Oo
    oo@Oo
     oooo

% java Art < sine60.grey
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
.O@@@@@@@@@@@@@@@@@@@@o..o@@@@@@@@@@@@@@@@@@@@Oo..o@@@@@@@@@
O.O@@@@@@@@@@@@@@@@@@oo@@O.@@@@@@@@@@@@@@@@@@@.O@@oo@@@@@@@@
@O.@@@@@@@@@@@@@@@@@O.@@@@oo@@@@@@@@@@@@@@@@@oo@@@@.O@@@@@@@
@@oo@@@@@@@@@@@@@@@@.O@@@@@.O@@@@@@@@@@@@@@@O.@@@@@O.@@@@@@@
@@@.@@@@@@@@@@@@@@@Oo@@@@@@Oo@@@@@@@@@@@@@@@oO@@@@@@oO@@@@@@
@@@oO@@@@@@@@@@@@@@.O@@@@@@@.@@@@@@@@@@@@@@@.@@@@@@@Oo@@@@@@
@@@O.@@@@@@@@@@@@@@.@@@@@@@@oO@@@@@@@@@@@@@oo@@@@@@@@.@@@@@@
@@@@.O@@@@@@@@@@@@Oo@@@@@@@@O.@@@@@@@@@@@@@.O@@@@@@@@oO@@@@@
@@@@oO@@@@@@@@@@@@.O@@@@@@@@@.@@@@@@@@@@@@@.@@@@@@@@@Oo@@@@@
@@@@Oo@@@@@@@@@@@@.@@@@@@@@@@oO@@@@@@@@@@@Oo@@@@@@@@@@.@@@@@
@@@@@.@@@@@@@@@@@Oo@@@@@@@@@@Oo@@@@@@@@@@oO@@@@@@@@@@@oO@@@@
@@@@@oO@@@@@@@@@@oO@@@@@@@@@@@.@@@@@@@@@@@.@@@@@@@@@@@Oo@@@@
@@@@@Oo@@@@@@@@@@.@@@@@@@@@@@@.O@@@@@@@@@O.@@@@@@@@@@@O.@@@@
@@@@@@.@@@@@@@@@O.@@@@@@@@@@@@oo@@@@@@@@oo@@@@@@@@@@@@@.O@@@
@@@@@@.O@@@@@@@@oo@@@@@@@@@@@@O.@@@@@@@@@.O@@@@@@@@@@@@oo@@@
@@@@@@oo@@@@@@@@.O@@@@@@@@@@@@@.O@@@@@@@O.@@@@@@@@@@@@@O.@@@
@@@@@@O.@@@@@@@O.@@@@@@@@@@@@@@oo@@@@@@@oo@@@@@@@@@@@@@@.O@@
@@@@@@@.O@@@@@@oO@@@@@@@@@@@@@@@.@@@@@@@.@@@@@@@@@@@@@@@oo@@
@@@@@@@O.@@@@@@.@@@@@@@@@@@@@@@@oO@@@@@Oo@@@@@@@@@@@@@@@@.@@
@@@@@@@@.O@@@@oo@@@@@@@@@@@@@@@@O.@@@@@.O@@@@@@@@@@@@@@@@oo@
@@@@@@@@@O.O@@o.@@@@@@@@@@@@@@@@@o.@@@.o@@@@@@@@@@@@@@@@@@.o
@@@@@@@@@O...o@@@@@@@@@@@@@@@@@@@@o...o@@@@@@@@@@@@@@@@@@@@o
@@@@@@@@@@OO@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@O@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

Part 2b. Enhance `Art.java` to support color inversion and/or conversion to black-and-white via a threshold value. This will be done via ***optional command line arguments***. NOTE: The program should behave as before if no command line arguments are given.

The first command line argument (if given), is an integer specifying whether to invert the greyscale values (0 = don't invert, 1 = invert). To calculate the inverted greyscale value use the following formula:

$$inverted = 1.0 - greyscale$$

The second command line argument (if given), is a floating-point threshold used to enable black-and-white only output. In black-and-white ASCII art, the only two valid output characters are space for black and @ for white. A pixel is black if its greyscale value is strictly less than the floating-point threshold value, a pixel is white otherwise. Both inversion and black-and-white output can be used at the same time. In this case, a pixel's greyscale value is first inverted and the inverted value determines if the pixel is displayed as black or white.

Here are some example runs:

```
% java Art 1 < 135.grey
@@@@@@@@@@
@.@OO@ooo@
@.@@O@o@@@
@.@OO@ooo@
@.@@O@@@o@
@.@OO@ooo@
@@@@@@@@@@
o o o o o
```

```
% java Art 1 0.8 < 135.grey
@@@@@@@@@@
@ @  @   @
@ @@ @ @@@
@ @  @   @
@ @@ @@@ @
@ @  @   @
@@@@@@@@@@
```

```
% java Art 0 2.0 < 135.grey








```

```
% java Art 0 < 135.grey

 O .. ooo
 O  . o
 O .. ooo
 O  .   o
 O .. ooo

o@o@o@o@o@
```

```
% java Art 0 0.2 < 135.grey

 @ @@ @@@
 @  @ @
 @ @@ @@@
 @  @   @
 @ @@ @@@

@@@@@@@@@@
```

```
% java Art 0 0.0 < 135.grey
@@@@@@@@@@
@@@@@@@@@@
@@@@@@@@@@
@@@@@@@@@@
@@@@@@@@@@
@@@@@@@@@@
@@@@@@@@@@
@@@@@@@@@@
```

```
% java Art 1 0.6 < 135.grey
@@@@@@@@@@
@ @@@@   @
@ @@@@ @@@
@ @@@@   @
@ @@@@@@ @
@ @@@@   @
@@@@@@@@@@
```

```
% java Art < 135.grey

 O .. ooo
 O  . o
 O .. ooo
 O  .   o
 O .. ooo

o@o@o@o@o@
```

% java Art 1 < sine60.grey

```
O.                    oOOo                    .oOOo
.O.                  oo  .O                   O.  oo
 .O                 .O    oo                 oo    O.
  oo                O.     O.               .O     .O
   O               .o      .o               o.      O.
   o.              O.       O               O       .O
   .O              O        o.             oo        O
    O.            .o        .O             O.        o.
    o.            O.         O             O         .o
    .o            O          o.           .o          O
     O           .o          .o           o.          o.
     o.          o.           O           O           .o
     .o          O            O.         .O           .O
      O         .O            oo         oo            O.
      O.        oo            .O         O.            oo
      oo        O.             O.       .O             .O
      .O       .O              oo       oo              O.
       O.      o.               O       O               oo
       .O      O                o.     oo                O
        O.    oo                .O     O.                oo
        .O.  oO                  oO   Oo                  Oo
         .OOOo                    oOOOo                    o
           ..                       .
```

% java Art 1 0.01 < sine60.grey

```
@@                    @@@@                     @@@@
@@@                  @@@@@@                   @@@@@@
@@@@                @@@@@@@@                 @@@@@@@@
 @@@               @@@@  @@@@               @@@@  @@@@
 @@@@              @@@    @@@               @@@    @@@
  @@@              @@@     @@@             @@@     @@@
  @@@@            @@@      @@@             @@@      @@@
   @@@            @@@      @@@@           @@@@      @@@
   @@@           @@@@       @@@           @@@       @@@
   @@@           @@@        @@@           @@@        @@@
    @@@          @@@        @@@           @@@        @@@
    @@@          @@@         @@@         @@@         @@@
    @@@         @@@          @@@         @@@          @@@
     @@@        @@@          @@@         @@@          @@@
     @@@        @@@          @@@@       @@@@          @@@
     @@@       @@@@           @@@       @@@           @@@@
     @@@@      @@@            @@@       @@@            @@@
      @@@      @@@            @@@@     @@@@            @@@
      @@@     @@@@             @@@     @@@             @@@@
       @@@    @@@              @@@@   @@@@              @@@
       @@@@  @@@@               @@@   @@@               @@@@
        @@@@@@@@                @@@@@@@@@                @@@
        @@@@@@@                  @@@@@@@                  @@
         @@@@@                    @@@@@                    @
```