COS 126 General Computer Science Spring 2013

Programming Practice Exam 2

This practice test has 2 programming parts. You have 90 minutes. The exam is open book, open note, and open booksite. You may use code from your programming assignments or the Introduction to Programming in Java booksite. No communication with any non-staff members is permitted. **If this were a real exam, you would have to write out and sign the Honor Code pledge before turning in the test, so you should do so to practice taking this time into account. Write out and sign the Honor Code pledge before turning in the test:**
"I pledge my honor that I have not violated the Honor Code during this examination."

———————————————

Signature

Problem	Score
Problem	Score
0	/1
1	/19
2	/10

Total	

Name:

NetID:

Precept:
(circle your precept)

P01	12:30	Dave Pritchard
P01A	12:30	Donna Gabai
P01B	12:30	Pawel Przytycki
P02	1:30	Tom Funkhouser
P02A	1:30	Allison Chaney
P02B	1:30	Pawel Przytycki
P02C	1:30	Vivek Pai
P02D	1:30	Siddhartha Chaudhuri
P03	2:30	Tom Funkhouser
P03A	2:30	Allison Chaney
P04	3:30	Vivek Pai
P04B	3:30	Shilpa Nadimpalli
P05	7:30	Shilpa Nadimpalli
P06	10am	Lennart Beringer
P07	1:30	Dave Pritchard
P07A	1:30	Kevin Lee
P07B	1:30	Siyu Liu
P08	12:30	Donna Gabai
P08A	12:30	Judi Israel
P09	11am	Judi Israel

0. **Cover Page** Write your name and netid, select your lecture and precept sections, and write out the honor code on the cover (now!) and sign it when you complete the test.

1. **Map of Points**

   Your first task is to write a Java class, `PUPoints`, whose `main` method plots a set of points onto StdDraw.

   Your drawing window must have a scale from 0 to 1 in both the X and Y directions. The program must take one command line argument, which is an image file that must be displayed as the background using this command: `StdDraw.picture(.5,.5,args[0],1,1);`

   You must read in pairs of latitude and longitude coordinates from standard input. There is no specified limit to the number of pairs in the set, but you are guaranteed that they will be pairs (that is, input will not end with a latitude coordinate).

   For each pair, you must convert each coordinate from degrees into map coordinates (between 0 and 1) so that you can plot them. To convert from degrees into map coordinates, use this equation:

   $mapCoord = \frac{degree - degree_{min}}{degree_{max} - degree_{min}}$

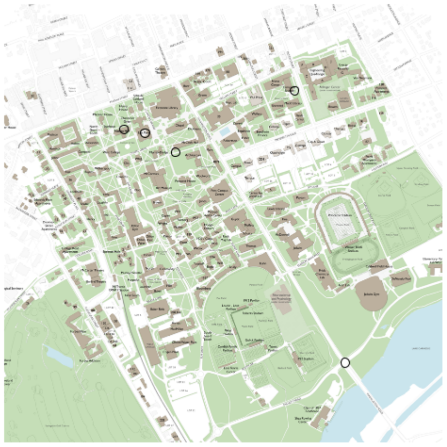
   The minima and maxima apply to the map provided. The minimum and maximum X (longitude) are -74.66443° and -74.64564°, respectively. The minimum and maximum Y (latitude) are 40.33855° and 40.35281°, respectively.

   Once you have completed the conversion for a pair, you must print that pair's map coordinates to standard output in the format shown in the examples (one latitude and longitude pair per line, separated by a space). You must also draw the point to the standard drawing interface as a black unfilled circle of radius .01.

Sample Run. Here is our input, output, and resulting drawing for three sample runs, the last of which is shown abbreviated. The full third dataset – real-world data courtesy of Leonardo Stedile '14, Spencer Tank '14, and Tiantian Zha '13 – is available on the Precepts page.

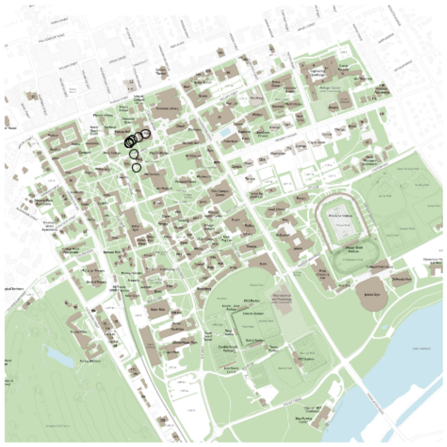
```
% more fivepoints.txt
40.35000 -74.65200
40.34800 -74.65710
40.34863 -74.65840
40.34875 -74.65931
40.34116 -74.64983

% java PUPoints map.png < fivepoints.txt
0.8029453015430142 0.6615220862159509
0.6626928471248922 0.39010111761563393
0.7068723702666181 0.320915380521391
0.7152875175318748 0.27248536455519407
0.18302945301572163 0.7770090473658801
```



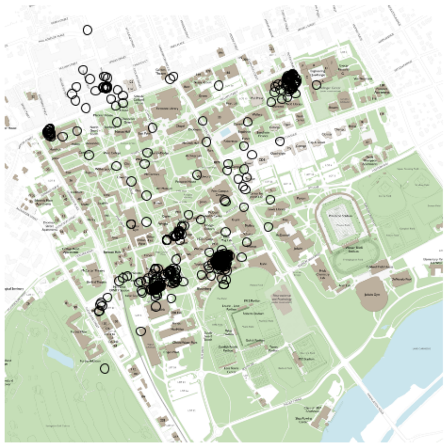
```
% more canon.txt
40.34863 -74.65840
40.34851 -74.65872
40.34842 -74.65896
40.34839 -74.65904
40.34832 -74.65910
40.34833 -74.65911
40.34833 -74.65912
40.34832 -74.65912
40.34832 -74.65912
40.34796 -74.65890
40.34752 -74.65880

% java PUPoints map.png < canon.txt
0.7068723702666181 0.320915380521391
0.6984572230013613 0.3038850452365496
0.6921458625525433 0.29111229377348574
0.6900420757366028 0.28685470995170814
0.6851332398319113 0.28366152208632034
0.6858345021037253 0.2831293241085036
0.6858345021037253 0.2825971261306869
0.6851332398319113 0.2825971261306869
0.6851332398319113 0.2825971261306869
0.6598877980366393 0.29430548163887355
0.6290322580648617 0.2996274614155283
```



```
% more stz.txt
40.34448 -74.65511
40.34448 -74.65511
40.34448 -74.65511
40.34455 -74.65504
40.34460 -74.65504
...
40.34359 -74.65794
40.34362 -74.65793
40.35022 -74.65109
40.34721 -74.65408
40.34747 -74.65418

% java PUPoints map.png < stz.txt
0.41584852734917654 0.49600851516788713
0.41584852734917654 0.49600851516788713
0.41584852734917654 0.49600851516788713
0.42075736325386814 0.49973390101109166
0.4242636746144333 0.49973390101109166
...
0.35343618513330904 0.34539648749339785
0.3555399719497478 0.3459286854712146
0.818373071528903 0.7099521021821478
0.6072931276296567 0.5508249068656159
0.6255259467042965 0.5455029270889612
```



2. **Heat Map** Your second task is to write a Java class, `PUHeatMap`, whose `main` method plots the points in a more advanced manner – creating a heatmap with a custom resolution.

   In addition to the image file, which unlike in the previous program is now the 2nd command line argument, this program takes another command line argument: N. Your drawing window must have a scale from 0 to N in both the X and Y directions.

   The input format is identical, as is the conversion to map coordinates, and the same pairing guarantee is still applicable. For this program, however, you must read all points and record their locations scaled onto an $N \times N$ grid. For example, the map coordinate $(.1, .2)$ should be tallied as being in grid cell $[floor(.1 * N)][floor(.2 * N)]$. ($floor$ just means use the integer part of the product and ignore the fractional part.)

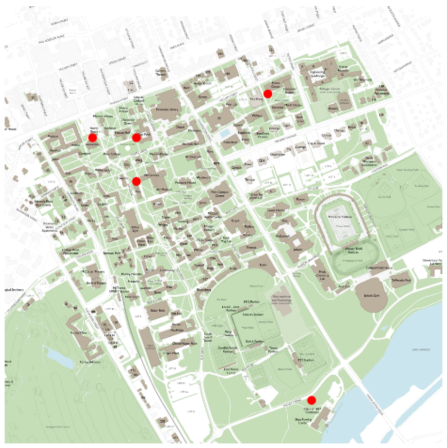
   Once you have read in all pairs of coordinates, you must scan through all grid cells and draw a point for each non-empty one to the standard drawing interface at its cell location (i.e. you must check N^2 cells, and if, e.g. cell [15][12] is not zero, draw a point at $(15, 12)$). The point should be drawn as a filled circle of radius $.01 * N$, with a color corresponding to the value of the cell:

```
< .1% of total points => StdDraw.BLUE
< .3% of total points => StdDraw.YELLOW
< .5% of total points => StdDraw.ORANGE
< .7% of total points => StdDraw.PINK
< .9% of total points => StdDraw.MAGENTA
>=.9% of total points => StdDraw.RED
```

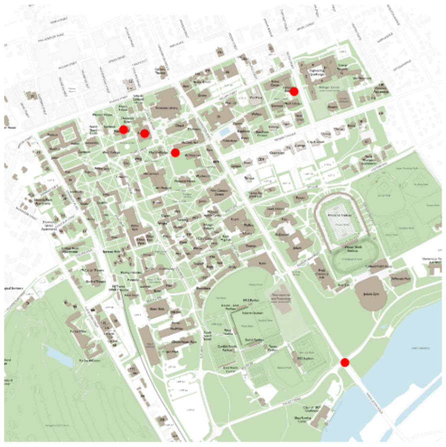
   Additionally, your program should print the total number of points and total number of non-empty cells, which corresponds to total coverage of the map, to standard output in the format shown in the sample runs.

   Sample Run. Here is our output and drawing for the same three sample files, using several values of N:

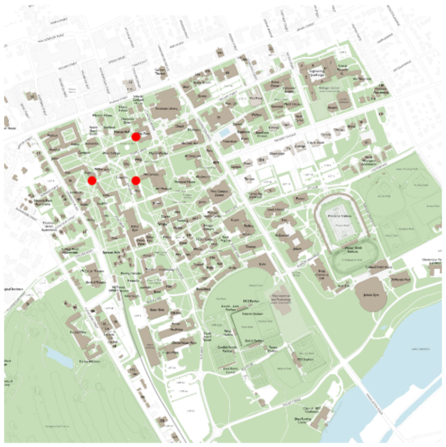
```
% java PUHeatMap 10 map.png < fivepoints.txt
Total points: 5
Filled cells: 5
```



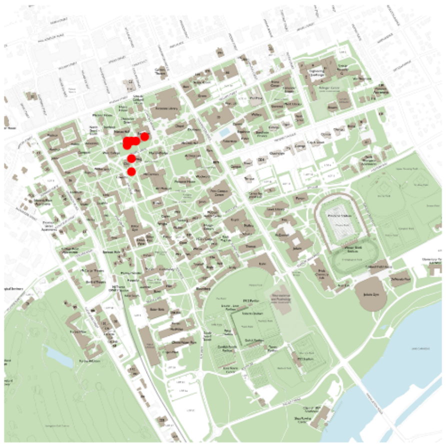
```
% java PUHeatMap 1000 map.png < fivepoints.txt
Total points: 5
Filled cells: 5
```



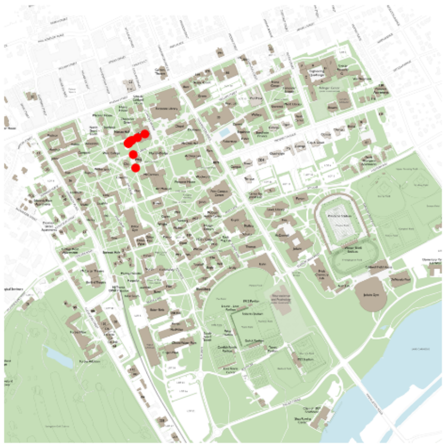
```
% java PUHeatMap 10 map.png < canon.txt
Total points: 11
Filled cells: 3
```



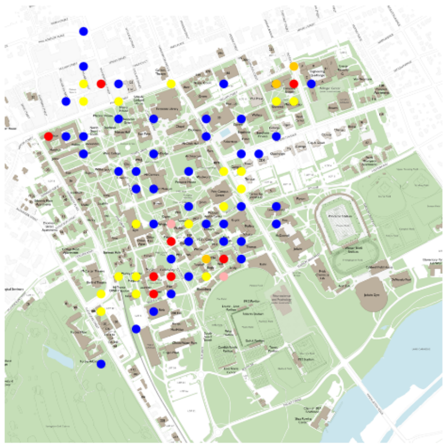
```
% java PUHeatMap 100 map.png < canon.txt
Total points: 11
Filled cells: 7
```



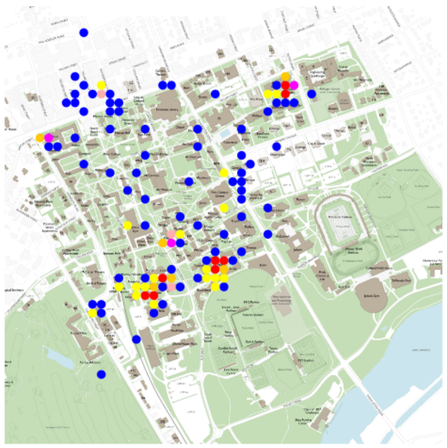
```
% java PUHeatMap 2000 map.png < canon.txt
Total points: 11
Filled cells: 10
```



```
% java PUHeatMap 25 map.png < stz.txt
Total points: 1492
Filled cells: 71
```



```
% java PUHeatMap 50 map.png < stz.txt
Total points: 1492
Filled cells: 110
```



```
% java PUHeatMap 100 map.png < stz.txt
Total points: 1492
Filled cells: 154
```