COS 126 **General Computer Science** **Spring 2014**

Programming Exam 1

This exam is semi-open: you may read the course website, the booksite, any direct links from those two, the textbook, any printed or written notes, and your past coursework on your computer. You may not use other websites. You must not post, write, or send information to anywhere. No other communication is permitted, except with course staff members. **Do not remove this exam from the exam room. Return it to course staff at the end of the exam.**

Upload your code to the dropbox. As with the COS 126 assignments, you can submit your code multiple times for testing, but only the last version will be graded.

Your code will be graded primarily on correctness. You will lose a substantial number of points if your program does not compile or has the wrong API. However, efficiency, clarity, and code style are also factors in your grade. Remember to include headers and comments in your code. Headers MUST include your name, netID and precept number.

Print your name, netID and precept number on this page (now), and write out and sign the Honor Code pledge before turning in this paper. It is a violation of the Honor Code to discuss this exam until everyone in the class has taken the exam. You have 90 minutes to complete the test.

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."

Pledge: ______________________________

____

Signature: ______________________

Name: ______________________

NetID: ______________________

Precept: ______________________

P01	12:30 TTh	David Pritchard
P01A	12:30 TTh	Donna Gabai
P01B	12:30 TTh	Aleksey Boyko
P02	1:30 TTh	Borislav Hristov
P02A	1:30 TTh	Terry Yannan Wang
P02B	1:30 TTh	Aleksey Boyko
P02C	1:30 TTh	Nanxi Kang
P02D	1:30 TTh	Xinyi Fan
P03	2:30 TTh	Borislav Hristov
P03A	2:30 TTh	Bebe Shi
P04	3:30 TTh	Kevin Lee
P04A	3:30 TTh	Victor Shaoqing Yang
P05	7:30 TTh	Kevin Lee
P06	10:00 WF	Mojgan Ghasemi
P07	11:00 WF	Mojgan Ghasemi
P08	12:30 WF	Donna Gabai
P08A	12:30 WF	Maia Ginsburg
P09	1:30 WF	David Pritchard
P09A	1:30 WF	Maia Ginsburg
P09B	1:30 WF	Judi Israel
P10	2:30 WF	Judi Israel

Problem	**Value**	**Score**
1	14	
2A	8	
2B	8	
Total	**30**	

As you are no doubt aware, it snows during the winter in Princeton. In this exam, you will write two programs dealing with snow statistics and simulations. Part 1 is described on this page. The second program is divided into part 2A and part 2B. We encourage you to proceed sequentially through part 1, part 2A, and part 2B. However, part 2A can be finished independently of part 1.

After completing each part and testing it on your own computer, you can test it online. Go to the Assignments page (refresh the page if necessary) and click on the Submit link for this exam.

Part 1: `SnowStats` (14 points)

In the first half of this exam, you will write a program that reads Princeton's daily snowfall and temperature data from standard input, and prints out the total number of days worth of data, the total snowfall, and the coldest temperature.

The input format is as follows. Each line contains a decimal number (representing the inches of snowfall on that day) and an integer (representing the temperature in degrees Fahrenheit on that day), separated by a space. For example, here are the contents of a text file called `snow.txt` with this format:

```
8.0 21
2.0 24
0.0 45
2.4 30
1.9 19
4.4 26
```

You can obtain the text file `snow.txt`, and other text files used by the dropbox script, at

`http://www.cs.princeton.edu/~cos126/docs/data/Snow/`

Your first program, `SnowStats.java`, should (in its `main()` method) read this data and output the number of days, total snowfall, and coldest temperature. Your output should match the following example. When we run

```
java-introcs SnowStats < snow.txt
```

the output should be

```
Number of days: 6
Total snow: 18.700000000000003
Coldest temperature: 19
```

Note that we have some rounding error in the total snowfall, which is fine; the second half of the exam avoids this problem by using formatted printing which limits the number of decimal places printed.

The file can have any number of days (lines of data); there will always be at least one day. You may assume that the daily temperature is never more than 200 degrees.

You can test your part 1 code online before starting part 2.

In part 2 of the exam, you will write a program `SnowMelt` that simulates how fast the snow will melt, depending on the level of salt used to help melt it. The API has three methods: `meltage()` and `printArray()` described below, and `main()` described in part 2B.

Part 2A: `SnowMelt` methods (8 points)

The first method that you must create for this part must have the signature

```
public static double meltage(double currSnow, int temp, int salt)
```

This method computes the following: if there is `currSnow` inches of snow on the ground, the temperature today is `temp`, and we are using a `salt` percent salt solution, how much of that snow will melt? It should return the number given by the formula

$$\min\{\text{currSnow}, (1 + \text{currSnow})^{(\text{temp}-32+2\times\text{salt})/18}\}$$

For example, if `currSnow` is 8.0, `temp` is 21, and `salt` is 10, then this formula evaluates to

$$\min\{8.0, (1+8.0)^{(21-32+2\times10)/18}\} = \min\{8.0, 9.0^{0.5}\} = \min\{8.0, 3.0\} = 3.0$$

You will need to use `Math.pow()` to compute this.

Once you've written your code, you can test it in DrJava in the *interactions pane* by typing in the command `SnowMelt.meltage(8.0, 21, 10)` (without a semicolon, then press Enter). It should print `3.0`.

The second method that you must create for this part must have the signature

```
public static void printArray(double[] arr)
```

It should perform formatted printing of each element of the array using the format specifier

```
StdOut.printf("%8.3f", ____);
```

where `____` is a `double` value you want to print. Naturally, to print all of the elements, you will need a loop. After all elements are printed, your method should print a newline.

Once completed, here is one way that you can perform a very basic test of your method: in DrJava's interactions pane, enter `SnowMelt.printArray(new double[3])`, which should cause

```
   0.000   0.000   0.000
```

to be printed.

You can test your part 2A code online before starting part 2B.

Part 2B: `SnowMelt` main (8 points)

You will now write a `main()` method for `SnowMelt` with the following behavior: it reads a data file from standard input, in the same format as part 1, and takes any number of integer command-line arguments, representing possible salt concentrations Princeton might use to treat its hallowed sidewalks. Your program must print a table representing the amount of snow left on the ground after each day, for each of the specified salt percentages. We begin with an example and then describe other important details further below on this page. Read it all before starting to code.

For example, the following command asks your program to simulate two possible scenarios, one using a 10% salt solution, and one using a 5% salt solution:

```
java-introcs SnowMelt 10 5 < snow.txt
```

It produces the following output; the left column is for a 10% salt solution, the right column is for a 5% salt solution, and each row gives the amount of snow left on the ground after each day.

```
5.000   7.115
3.000   7.822
0.000   0.000
0.000   0.677
0.387   1.769
0.869   4.620
```

We now explain how you should generate this table. Each day, your simulation should (a) increase the amount of snow on the ground by the new snowfall, (b) call `meltage()` on the new total snow, and (c) reduce the amount of snow on the ground by the meltage. The snow left at the end of one day remains there at the start of the next day.

For example, consider a 10% salt solution, using the first two days of `snow.txt`:

```
8.0 21
2.0 24
...
```

1. On the first day, 8.0 inches of snow falls. Of this snow, `meltage(8.0, 21, 10)` will melt, which is 3.0 inches (this example was given in part 2A). So after one day, 8.0 − 3.0 = **5.0** inches of snow remains on the ground.

2. On the second day, 2.0 more inches of snow falls, *which when added to the 5.0 inches already on the ground*, gives a total of 7.0 inches. Of this snow, `meltage(7.0, 24, 10)` inches will melt, which is 4.0. So after two days, 7.0 − 4.0 = **3.0** inches of snow remains on the ground.

Thus, the first column of the sample output above starts with `5.000` and `3.000` in the top rows.

Like the example above, assume that before the first day, there is no snow on the ground.

Use an array of `double` values to simulate the remaining snow under the different salt concentrations. Print each row using `printArray`.

If you are unable to complete the program for multiple command-line arguments, we will award partial credit for a solution that works when it takes exactly one command-line argument.

Remember to submit your completed code online and to use the "Check All Submitted Files" button.