Contest Problems
Philadelphia Classic, Fall 2014
Hosted by the Dining Philosophers
University of Pennsylvania

dining philosophers

UNIVERSITY OF PENNSYLVANIA | COMPUTER SCIENCE CLUB

Rules and Information

This document includes 15 problems. Novice teams do problems 1-10; standard teams do problems 6-15.

Any team which submits a correct solution for any of problems 1-5 will be assumed to be a novice team. **If you are not a novice team, please skip problems 1-5.**

Problems 1-5 are easier than problems 6-10, which are easier than problems 11-15. These problems are correspondingly labeled “Novice”, “Intermediate”, and “Advanced.” Order does not otherwise relate to difficulty, except that problem 15 is the hardest.

You may use the Internet only for submitting your solutions, reading Javadocs, and referring to any documents we link you to. You **may not** use the Internet for things like StackOverflow, Google, or outside communication.

Do not modify any of the given methods or method headers in our stub files! They are all specified in the desired format. You may add class fields, helper methods, etc as you like, but modifying given parts will cause your code to fail in our testers.

There is no penalty for incorrect submissions. You will receive 1 point per problem solved. A team’s number of incorrect submissions will be used only as a tiebreaker.

Some problems use Java’s “long” type; if you are unfamiliar with them, they’re like an “int”, but with a (much) bigger upper bound, and you have to add “L” to the end of an explicit value assignment:

```
    long myLong = 1000000000000L;
```

Otherwise, the “long” type functions just like the “int” type.

1. Novice Problem - Penguin Mathematics

There is a penguin named Clever Hans[1] that is reputed to know how to do basic arithmetic with an abacus! A challenger will present Hans with two abaci (the plural form of abacus), each of which represents one of two numbers to add together, and Hans will move the beads on a third abacus to equal the sum of the two numbers.

As a reminder, each row of an abacus represents a digit of a number. The first row represents the ones digit, and the second row represents the tens digit, for example. The number of abacus beads shifted to the left hand side of the row is the value of the digit in that spot. For example, if three (out of nine) beads were on the left hand side of the third row (the hundreds row), that would mean the hundreds digit is 3.

For the sake of this problem, an abacus is represented as String of a series of nine asterisks and a space, which is comma-delimited. All asterisks to the left of the space represent the count at that digit of the number (the moved beads), and all asterisks to the right of the space are unmoved beads. Asterisk-space combinations toward the beginning correspond to "higher" digits than asterisk-space combinations toward the end.

You will receive a semicolon-delimited String of two abaci as input. You should output the abacus that Hans would output. Please do not include any extra leading zeros.

You may assume that all inputs are valid.

Sample Input:	**Parsed Input:**	**Sample Output:**
`***** ****;*** *******`	5 + 3 (= 8)	`******** *`
` *********,**** *****;****** ****`	4 + 6 (= 10)	`* ********, *********`
`* ********, *********,********* ;****** ***`	109 + 6 (= 115)	`* ********,* ********,***** ****`

[1] http://en.wikipedia.org/wiki/Clever_Hans

2. Novice Problem - Parental Penguins Poorly Protect Pre-Kindergarten Pupils

The penguins have decided that forming a day-care will be easier than trying to do the whole stand around in the freezing cold all winter thing. In order to keep from losing any chicks the organizers of the day-care paint numbers on each of them so that it's easy to tell which one is missing. Because of the large volume of chicks, periodically going through and checking each number is a very tedious task. The penguins decide to mitigate this by writing a program to determine which penguin is missing given a list of all of the numbers the day-care organizers read off of the chicks.

The stub function will have two arguments, a list of integers representing the numbers of the chicks and the total number of chicks. The output of the function is an integer representing the number of the penguin not present, or -1 if the day-care has not lost any. You can assume that there will be at least one penguin present and that penguin indexing begins at 1.

Sample Input:	**Sample Output:**
1). [7, 1, 5 ,4, 3, 6], 7	2
2). [5, 3, 4, 1, 2], 5	-1
3). [3, 2, 4], 4	1

3. Novice Problem - Phoney Numbers

To cut down on the amount of cross continent pilgrimages, the penguins have created a simple telephone system spanning Antarctica. This has revolutionized penguin society much in the same way that phones revolutionized human culture. The general opinion among penguins is that phones just aren't as personal as handwritten letters and that the whole thing was a bad idea. Another consequence of this is that every penguin form asks for a phone number for some reason. Since no one came up with a standard way of formatting numbers, there are now a huge amount of phone numbers written down that need to be turned into some standard format.

The PISO (Penguin International Organization for Standardization) has decreed that the international standard format for phone numbers is (X) XX-XX, where X can be any digit from 0 to 9. Note: There is a single space after the close parenthesis. Your job is to take a string and return the phone number contained in it (if any) in standard format. The formats that people have been using are as follows:

1. X-XX-XX
2. X XX-XX
3. (X)-XX-XX
4. (X) XX-XX

The stub function will have a single input which is a string that will contain at most one phone number in one of the specified formats. The phone number will not have any extra whitespace inside it (other than the one space in formats 2 and 4) but could have other characters surrounding it (possibly not whitespace!). You should return the phone number contained in the string in standard format or return null if it wasn't found.

Sample Input	**Sample Output**
1-23-45	(1) 23-45
My phone number is (5)-44-27	(5) 44-27
4 32-56howdoitype	(4) 32-56

4. Novice Problem - Not Very Russian Nesting Penguins

Our intrepid group of penguins are reaching that age where everyone is starting to build nests in anticipation of hatching and raising chicks. To make the task of building a bunch of nests less dull a group of penguins decide to work as a team in order to make it more fun. Each penguin individually knows how fast they can build a nest in terms of some number of nests built in some number of hours but they would like to know how fast their combined work rate is. Since these penguins are versed in programming they decide to write some code that will compute the total work rate in nests per hour.

The stub function will have two arguments, both of which are int arrays. The first array is a list of the number of nests each penguin can build in some time period. The second array specifies that time period for each penguin. Thus penguin with index 3 can build <nests[3]> nests in <hours[3]> hours. Your method will return a double indicating how fast the group of penguins can build nests in nests per hour. You can assume that the arrays will always be the same size and that they will not have size 0.

Sample Input	**Sample Output**
[3, 2], [2, 1]	3.5
[1, 2, 3], [1, 2, 3]	3
[4, 4, 4, 4], [2, 4, 2, 1]	9

5. Novice Problem- Penguin Building

The penguins are learning to build bridges to see their friends in Madagascar. Despite being able to build rockets, they are terrible at building bridges. They have trouble building triangular pieces to make a strong bridge. Whenever they randomly pick 3 boards, the lengths never seem make a triangle. Your job is to help the penguins determine if their 3 boards can make a triangle for their bridge.

The stub function will have three arguments, each one an int representing the length of one of the boards. Your method should return true if the lengths can be assembled to form a valid triangle, and false otherwise.

Sample Input	**Sample Output**
1, 2, 3	false
3, 4, 5	true
3, 9, 7	true

6. Intermediate Problem - A Penguin Jumped Over the Moon

In an attempt to escape from the effects of global warming, the penguins are assembling a rudimentary space program. At the current stage of development, they are attempting to launch a “penguin-ified” rocket and then have a capsule return the penguin safely to earth. Unfortunately, they have to keep changing their launch and landing sites to adapt to the ever-changing Antarctic region. They know that the capsule will land within a certain distance away from the launch site, determined by launch conditions. They also are able to set up a rectangular site in which it is safe to land. The penguins would like to know given their launch site, landing radius, and rectangular landing zone if there is any possibility that the capsule will land in the safe zone.

The stub function will have 7 arguments. The first two are the x and y coordinates of the launch site. The third is the radius within which the capsule will land. The fourth and fifth are the x and y coordinates of the upper left hand corner of the rectangle and the sixth and seventh are the width and height. You should output a boolean that is true if there is a chance of a safe landing or false if there is no chance.

Note: If the landing circle is tangent to the safe area then there is a chance of landing safely.

Sample Input:	**Sample Output:**
1). 30, 40, 30, 0, 10, 40, 40	true
2). 10, 20, 10, 45, 20, 10, 10	false
3). 0, 10, 5, 0, 6, 6, 5	true

7. Intermediate Problem - Penguin Information

Some people think Penguins are simple and uninteresting birds, but they couldn't be more wrong. Each penguin is born special and unique! Penguins have names, favorite foods, weights, favorite musical artists, and, favorite poets.

We have too much interesting information about our penguin friends, but we need your help organizing! Given a list of penguins and something to sort the penguins by, print out the penguins in order.

We define a Penguin object for you in PenguinInfo.java. A Penguin object has the following methods:

Method	**Output Type**	**Output Description**
getName()	String	This is the name of the penguin.
getFavoriteFood()	String	This is the penguin's favorite cuisine.
getWeight()	double	This is the penguin's weight in pounds.
getHeight()	int	This is the penguin's height in inches.
getMusicalArtist()	String	This is the penguin's favorite musical artist.
getPoet()	String	This is the penguin's favorite poet.

Input:

The method has the following parameters:

- List penguins - the list of Penguin objects to sort. Remember that penguins are unique, so no two penguins will have the same information.
- String sort - the field to sort the penguins by. Sort will only have one of the following values.
  - "FOOD"
    - Sort the penguins by favorite food in ascending alphabetical order.
  - "WEIGHT"
    - Sort the penguins by weight in ascending numerical order.
  - "HEIGHT"

        - Sort the penguins by height in ascending numerical order.
    - “MUSIC”
        - Sort the penguins by favorite musical artist in ascending alphabetical order.
    - “POET”
        - Sort the penguins by favorite poet in ascending alphabetical order.

Output:

You will return a string representing the list of penguins in sorted order. Use the toString method supplied to you to convert the penguins to strings. They should be separated by spaces with no leading or trailing whitespace.

Sample Input:	**Sample Output:**
List<Penguin> penguins - {Penguin(“Jack”,”Pizza”,50.43,40.32,”Kanye West”,”John Keats”), Penguin(“Peter”,”Sushi”,45.345,32.1234,”Jay-Z”,”Sylvia Plath”) } String sort - “FOOD”	Jack Peter
List<Penguin> penguins - {Penguin(“Jack”,”Pizza”,50.43,40.32,”Kanye West”,”John Keats”), Penguin(“Peter”,”Sushi”,45.345,32.1234,”Jay-Z”,”Sylvia Plath”) } String sort - “WEIGHT”	Peter Jack

8. Intermediate Problem - Nest and Breakfast

Penguin Joe and his wife Penguinette Joelle have decided to open a small "nest-n-breakfast". After getting everything set up, they posted a calendar where the other penguins could sign up for dates during which they wanted to stay. Much to their surprise, the N and B was a huge hit, and they quickly became totally booked. They tried to keep up with the demand, but sadly it turns out that they need time in between bookings to clean the place; penguins are extremely messy house guests. Joe and Joelle decide that they will just have to tell some of the people that they can't honor their reservations, but they still want to keep as much time booked as possible. They need your help building a program that will decide which reservations they should keep.

The input to the stub function will consist of an array of integers that represents the duration of each booked stay (in days) in chronological order. There is no need to include any other information since each new reservation begins immediately after the previous one ends. You should return the maximum number of days Joe and Joelle can have guests while never having two stays back to back.

Sample Input	**Sample Output**
5, 1, 5	10
10, 4, 5, 9, 1	19
1, 5, 3	5

9. Intermediate Problem - Penguin Concert Seating

Phish[1] is coming to town, and the penguins are going to see them in concert. The concert venue is an isosceles triangular field, with the stage at the point between the two sides of equal length. It's cold and rainy outside, so the penguins will be huddled together tightly for warmth, and there will be a tent over the entire field to keep the rain out.

Unfortunately, holding up the tent requires some pillars which block the view. After the setup crew installs the tent, we need to figure out how many penguins can come see the show. Fortunately, penguins huddle very tightly[2] - they fit 21 per square meter!

The pillars are line segments parallel to the base of the triangle. Their position is represented by three coordinates (x1, x2, y), with the origin at the bottom-left corner of the triangle. All pillars are contained entirely within the triangle, and there are no "overlapping" pillars; any spot where the view is blocked by a pillar will have the view blocked by only one pillar.

Given the dimensions of the triangular field (base and height) and a list containing pillar coordinates, compute how many penguins will be able to see the show without having their view blocked by a pillar. In other words, find the total area within the triangle such that a straight line between the apex (stage) and that point will not intersect any pillars, and then account for penguin density. Note that the solution should be rounded down to an integer, since Phish would rather not have fractional penguins milling around their show.

All dimensions and coordinates are in meters.

Sample Input:	**Sample Output:**
10, 10, [4, 6, 5]	735
10, 5, [1, 9, 1]	336
10, 5, [4, 5, 4, 5, 8, 2]	105

[1]: http://en.wikipedia.org/wiki/Phish
[2]: http://www.plosone.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0020260

10. Intermediate Problem - Penguin by North Penguin

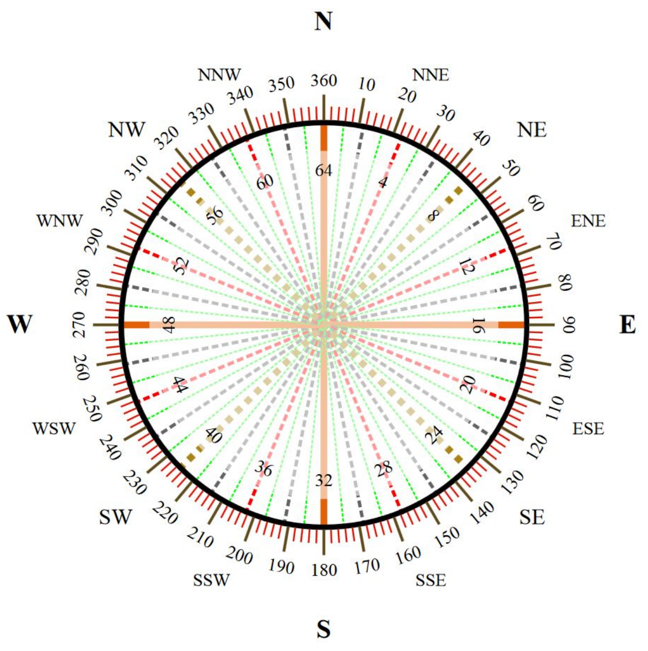
Our intrepid penguins have decided to set off on an expedition to explore the area surrounding the South Pole. They have figured out that the best way to start exploring is to start at the Pole, have each time pick a direction and set off in a straight line. The director of penguin exploration (DPE) has sent out a list giving each penguin a direction given in degrees but unfortunately the penguins are much more at ease with directions given in ordinal form. That is instead of a 45° heading the penguins would rather hear NE. In order to deal with this problem the DPE has contracted you to write a program that will convert directions given in degree form into ordinal form. Since not all directions can be converted into ordinal form the DPE will also provide a tolerance that the ordinal form can be within.

The rules for determining the ordinal form of a direction are as follows:

1. All directions have a either a E or W as the last character unless it is simply N or S
2. All directions are made of either E or W and either N or S
3. A character that is *i* characters away from the last one moves the total 45/2^i *in the direction of that direction's degree number.*
4. The total starts at the last character's degree number (ie either 90 or 270).

The picture to the left should help clarify:

http://blog.weatherflow.com/degrees-of-wind-direction-along-south-carolina-onshore-vs-offshore/



The stub function will have two arguments. A double representing the ordinal direction the expedition is heading in and another double that represents the the tolerance within which the degree representation of your ordinal representation must fall within. You should compute the shortest sequence that falls within the tolerance.

Sample Input	**Sample Output**
22.5, 0.1	NNE
260, 5.0	WWSW
182, 2.0	S

11. Advanced Problem - Penguins versus Robots

The penguins are trying to get through a long valley guarded by evil robots! Each robot has a detection range, and any penguin which walks within that range of a robot will immediately be attacked. Luckily, the penguins are really good at hacking, so they can spend time to deactivate the robots one by one. However, they don't want to raise the suspicions of the robot overmind by deactivating too many robots. Thus, help them decide the least number of robots they need to deactivate so as to create a path for them to slip through!

The valley is represented by a rectangle on the xy-plane with bottom-left corner (0, 0) and top-right corner (**valleyWidth**, **valleyHeight**). The penguins start at the left end of the valley (at x = 0) and are trying to get to the right end (at x = **valleyWidth**). The lines from (0, 0) to (**valleyWidth**, 0) and from (0, **valleyHeight**) to (**valleyWidth**, **valleyHeight**) are two high walls which cannot be crossed.

You are given:

- a long **valleyWidth**
- a long **valleyHeight**

Also, for each robot, you are given:

- a long **xPosition**
- a long **yPosition**
- a long **detectionRange**

From this, calculate the minimum number of robots which need to be deactivated so as to allow the penguins to slip through

Sample Input	**Sample Output**	**Explanation**

20 4
1
10 2 2
	1	There is one robot guarding the entire valley, standing at (10, 2) and with detection range 2 units, which needs to be removed to pass

20 4
1
10 1 2
	0	There is a gap for the penguins to slip through, so no robot needs to be removed

20 4
3
10 1 2
11 1 2
10 3 4
	1	Only the third robot needs to be removed for a gap to be created

12. Advanced Problem - Penguin Procrastination

It turns out that penguins are even worse procrastinators than high school (and college) students. They love 2048[1], and will waste countless hours playing the game. Unfortunately, these penguins are not really that good at math (go figure, they're too busy playing to study). Thus, they sometimes will soldier on, even when it is clear they will be unable to win the game. Your job is to help them (in a limited case) to determine when a game is fruitless.

You will be given a 2-dimensional integer array sized 4x4, filled with numbers representing the current state of a 2048 game (you can be guaranteed this is a valid board). Your job is to determine whether, starting from this position, the Penguins could win the game in 7 moves (i.e. 7 keystrokes) or fewer (ignoring the addition of other tiles).

You should return true if a win is possible, false otherwise.

Sample Input

16	16	32	64
0	0	0	128
0	0	0	256
0	0	1024	512

Sample Output	true
Explanation	A possible winning move sequence is right, right, right, down, down, down, left

Sample Input

2	2	4	8
0	0	0	16
0	0	0	32

0	0	128	64

Sample Output	false
Explanation	Not enough pieces on the board to get 2048

Sample Input

0	0	0	0
0	0	0	0
0	0	0	0
512	1024	512	0

Sample Output	false
Explanation	While there are enough pieces on the board to make 2048, you cannot align them properly without additional tiles.

Sample Input

8	8	16	32
0	0	0	64
0	0	0	128
0	1024	512	256

Sample Output	false
Explanation	While it is possible to win this game, it will take more than the allotted 7 moves

[1] http://gabrielecirulli.github.io/2048/

13. Advanced Problem - Penguin Iceberg Hopping

Iceberg hopping is a popular pastime amongst young and energetic penguins. There are some icebergs floating in around in the water. However, the penguins play by strange and unusual rules: each iceberg has a specific next iceberg that can be hopped to, or else the hopping penguin loses. For example, the next iceberg from iceberg 2 might be iceberg 4, so any penguin who hops onto iceberg 2 must hop to iceberg 4 next. Note that each iceberg has only one “next iceberg”, but it can be possible that many icebergs all point to one iceberg as their “next iceberg”.

A bunch of lazy penguins is getting tired of all the hopping. Each of them is currently standing on some iceberg, and wants to hop some number of times. Help them find out what iceberg each of them will end up on, so that they won’t have to do all the hopping to find out.

Your task is to calculate the ending position for each penguin assuming they follow all the rules. You are given:

- a long[] **nextIceberg**, an array containing the next iceberg that should be hopped to from each iceberg. In other words, a penguin should hop to iceberg **nextIceberg[i]** from iceberg **i**
- a long[] **startingPositions**, an array containing the starting iceberg of each penguin.
- a long[] **numHops**, an array containing the number of hops each penguin wants to hop.

From this, return a long[] **endingPositions**, an array containing the ending position of each penguin.

Sample Input	**Sample Output**	**Explanation**

3 3
1 2 0
0 1 2
3 6 9
	[0, 1, 2]	**nextIceberg** = {1,2,0}
		startingPositions = {0,1,2}
		numHops = {3,6,9}
		
		Hopping 3 times from any iceberg in the given configuration will bring you back to your starting position.

4 5
1 0 0 0
0 1 2 3 3
3 1 3 3 7
	[1, 0, 0, 0, 0]	**nextIceberg** = {1,0,0,0}
		startingPositions = {0,1,2,3,3}
		numHops = {3,1,3,3,7}
		
		As an example, the sequence of icebergs taken by the 5th penguin in 7 hops would be:
		3->0->1->0->1->0->1->0, ending on 0.

14. Advanced Problem - Penguin Fish Catching

The penguins are playing a game with fishes. One penguin stands on a piece of ice, represented by a straight line, while the other penguins start tossing fish to him. However, their aim is really bad, so the fish don't all land near the penguin catching them. Any fish that the penguin misses and hits the ice flops into the sea and is lost forever. To catch a fish, a penguin must be standing on exactly the landing position of the fish at exactly the landing time of it. For example, if a fish has landing time 5 and landing position 3, the penguin must be standing on position 3 exactly at time 5 to catch the fish. Of course, the penguin may waddle slower than (but not faster than!) his waddling speed if he wishes to, or even stop completely, if that will help him to catch more fish.

Your job is to help the penguin catch as many fish as possible. You are given:

- a long **startingPosition**
- a long **waddlingSpeed**
- a long[] **landingTimes**
- a long[] **landingPositions**

From this, calculate the maximum number of fish it is possible for the penguin to catch.

For the sample input given below, the first number represents the starting position and the second number represents the waddling speed (in feet per second). Then, the first number of each Fish represents the landing time (in seconds after the game starts), and the second number represents the landing position of the fish.

Sample Input	**Sample Output**	**Explanation**

5 1
3
1 2 3
4 7 2
	2	**startingPosition** = 5
		waddlingSpeed = 1
		landingTimes = {1, 2, 3}
		landingPositions = {4, 7, 2}
		
		The penguin can catch the first and third fish, but the second is too far away to be caught.

5 5
1
2
100
	0	**startingPosition** = 5
		waddlingSpeed = 5
		landingTimes = {2}
		landingPositions = {100}
		
		The fish is too far away to be caught.

15. Advanced Problem - Penguin sums

A group of mathematically inclined penguins are doing a programming contest, and they need your help with the last problem. There are N penguins, labeled from 1 to N, and each of them has a favorite number that is the largest integer which divides both N and their own label (in other words, the g.c.d. of N and their label). Help the penguins find the sum of all their favorite numbers. Since this may be very large, you will also be given a long M, and your job is to output the remainder when this sum is divided by M.

You will be given:

- a long **N**, indicating the number of penguins.
- a long **M**, indicating the mod to be used.

From these, print out the sum of all the penguins' favorite numbers mod **M.**

Sample Input	**Sample Output**	**Explanation**
10 10000	27	1 + 2 + 1 + 2 + 5 + 2 + 1 + 2 + 1 + 10 = 27 27 (mod 10000) = 27
8 10	0	1 + 2 + 1 + 4 + 1 + 2 + 1 + 8 = 20 20 (mod 10) = 0