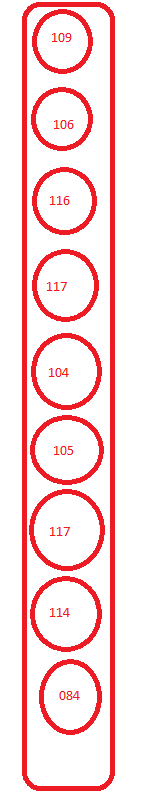
Programming contest:

1. Professor Moriarty is obsessed with Fibonacci series and sherlock knows it. Professor Moriarty has send sherlock a tube containing red balls. Each of them are numbered. Now sherlock has reasons to believe if he subtracts fibonacci numbers each time he pops a ball from the tube and turns it to ascii value he will get to know what Moriarty is saying. Now Sherlock came to you. Please help sherlock to unlock the message.

As example first ball that pops is 109. First Fibonacci number is 0. So (109 – 0) = 109. 109 is the ascii value of ‘m’. So the first letter is m.Similarly second number will be (106-1) = 105.



2. John is working in a diamond mine, trying to extract the highest number of diamond "<>". He must exclude all sand particles found "." in this process and after a diamond can be extracted, new diamonds can be formed. If he has as an input. <... << .. >> ....> .... >>>. three diamonds are formed. The first is taken from <..> resulting. <... <> ....> .... >>>. The second diamond is then removed, leaving. <.......> .... >>>. The third diamond is then removed, leaving at the end ..... >>>. without the possibility of extracting new diamonds.

**Input**

String including "<" ,">" and "."

**Output**

You must print the amount of diamonds that can be extrated in each test case.

| **Input Sample** | **Output Sample** |
| --- | --- |
| <..><.<..>> <<<..<......<<<<....> | 3 1 |

3. a) Write down the insertion method of a Binary Search Tree recursively in such a way so that whenever it finds a duplicate value to insert it goes to the left of the tree

/\*\*

\* @param key the key to insert.

\* @param root the root to this tree which may have changed.

\* @return reference to the root, which may have changed.

\*/

public Node insert (Object key, Node root) {

//TO DO

}  
b) Write down a print method that prints the odd elements in the binary search tree in descending order.

public void printOdd (Node root) {

//TO DO

}

4. Given a non-empty array, return true if there is a place to split the array so that the sum of the numbers on one side is equal to the sum of the numbers on the other side.

canBalance([1, 1, 1, 2, 1]) → true  
canBalance([2, 1, 1, 2, 1]) → false  
canBalance([10, 10]) → true

5. Given n>=0, create an array length n\*n with the following pattern, shown here for n=3 : {0, 0, 1,    0, 2, 1,    3, 2, 1} (spaces added to show the 3 groups).

squareUp(3) → [0, 0, 1, 0, 2, 1, 3, 2, 1]  
squareUp(2) → [0, 1, 2, 1]  
squareUp(4) → [0, 0, 0, 1, 0, 0, 2, 1, 0, 3, 2, 1, 4, 3, 2, 1]

6. Consider the leftmost and righmost appearances of some value in an array. We'll say that the "span" is the number of elements between the two inclusive. A single value has a span of 1. Returns the largest span found in the given array. (Efficiency is not a priority.)

maxSpan([1, 2, 1, 1, 3]) → 4  
maxSpan([1, 4, 2, 1, 4, 1, 4]) → 6  
maxSpan([1, 4, 2, 1, 4, 4, 4]) → 6

7. We'll say that a "mirror" section in an array is a group of contiguous elements such that somewhere in the array, the same group appears in reverse order. For example, the largest mirror section in {1, 2, 3, 8, 9, 3, 2, 1} is length 3 (the {1, 2, 3} part). Return the size of the largest mirror section found in the given array.

maxMirror([1, 2, 3, 8, 9, 3, 2, 1]) → 3  
maxMirror([1, 2, 1, 4]) → 1  
maxMirror([7, 1, 2, 9, 7, 2, 1]) → 2

1. Given the reference of a singly linked list containing integers, rearrange the values in the list so that the odd numbers appear first and then the even numbers. Make the initial list by passing an array in the constructor.
2. Given the reference of a doubly linear linked list, reverse the alternate nodes and append them to the end of the list. Throw EmptyListException if list is empty.

Input List: 1->2->3->4->5->6

Output List: 1->3->5->6->4->2

Input List: 12->14->16->18->20

Output List: 12->16->20->18->14

1. Print the following recursively

for input: 5

a)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
|  | 1 | 2 | 3 | 4 |
|  |  | 1 | 2 | 3 |
|  |  |  | 1 | 2 |
|  |  |  |  | 1 |

b)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | 1 |
|  |  |  | 1 | 2 |
|  |  | 1 | 2 | 3 |
|  | 1 | 2 | 3 | 4 |
| 1 | 2 | 3 | 4 | 5 |

1. Suppose you have been hired to develop a musical chair game. In this game there will be 7 participants and all of them will be moving clockwise around a set of 7 chairs organized in circular manner while a music will be played in the background. You will control the music using random numbers between 0-3. If the generated random number is 1, you will stop the music and if the number of participants who are still in the game is n, the participant at position (n/2) will be eliminated. Each time a participant is eliminated, a chair will be removed and you have to print the player names who are still in the game. The game will end when there will be only one participant left. At the end of the game, display the name of the winner.

[Hint: This is the code to generate a random number between 0(inclusive) to 3(inclusive):

import java.util.Random;

Random rand = new Random();

int n = rand.nextInt(4);

]