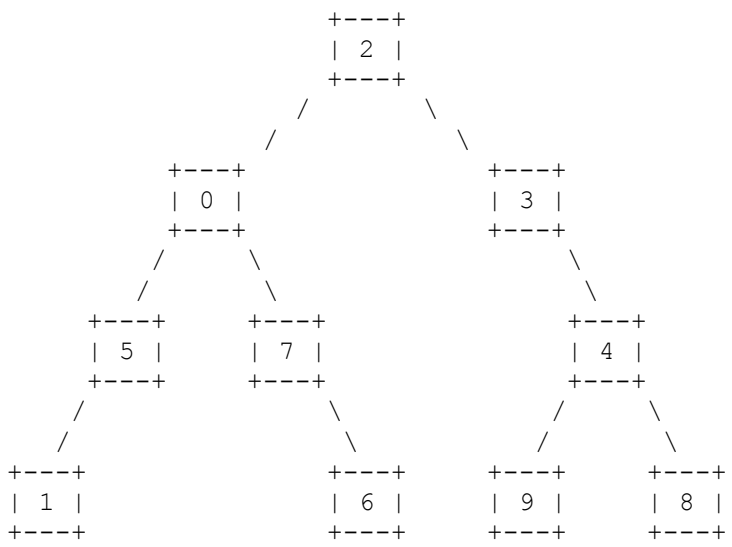
1. Binary Tree Traversals, 6 points. Consider the following tree.



Fill in each of the traversals below:

Preorder traversal _____

Inorder traversal _____

Postorder traversal _____

2. Recursive Programming, 9 points. Write a recursive method called weave that takes two integers as parameters and that returns the result of weaving their digits together to form a single integer. Two numbers x and y that have the same number of digits are weaved together as follows. The first digit of x and the first digit of y become the first two digits of the result in that order. The next pair of digits in the result comes from the second digit of x followed by the second digit of y. And so on.

   For example, consider weaving 128 with 394. The first two digits of the result are 13 because those are the first digits of the two numbers. The next two digits in the result are 29 and the third pair of digits is 84. So the overall result is 132984. Thus, the call weave(128, 394) should return 132984. Notice that the order of the arguments is important. The call weave(394, 128) would return 319248.

   If one of the numbers has more digits than the other, you should imagine that leading zeros are used to make the numbers have equal length. For example, weave(2384, 12) should return 20308142 (as if it were a call on weave(2384, 0012)). Similarly, weave(9, 318) should return 30198 (as if it were a call on weave(009, 318)). The table below includes several examples.

Method Call	Result	Method Call	Result
weave(8, 5)	85	weave(7, 0)	70
weave(5, 8)	58	weave(0, 7)	7
weave(42, 95)	4925	weave(4723, 9815)	49782135
weave(42, 7596)	7054926	weave(0, 0)	0
weave(7596, 42)	70509462	weave(444, 318)	434148

   The method should throw an IllegalArgumentException if either parameter is negative. You are not allowed to construct any structured objects to solve this problem (no string, array, ArrayList, StringBuilder, Scanner, etc) and you may not use a while loop, for loop or do/while loop to solve this problem; you must use recursion.

3. Details of inheritance, 10 points. Assuming that the following classes have been defined:

```
public class Green extends Red {
    public void method1() {
        System.out.println("Green 1");
    }

    public void method3() {
        System.out.println("Green 3");
    }
}

public class White extends Red {
    public void method2() {
        System.out.println("White 2");
    }

    public void method3() {
        System.out.println("White 3");
    }
}

public class Blue {
    public void method1() {
        System.out.println("Blue 1");
        method2();
    }

    public void method2() {
        System.out.println("Blue 2");
    }
}

public class Red extends Blue {
    public void method2() {
        System.out.println("Red 2");
        super.method2();
    }
}
```

And assuming the following variables have been defined:

```
Blue var1 = new Blue();
Green var2 = new Green();
Object var3 = new White();
Red var4 = new Green();
Blue var5 = new Red();
Blue var6 = new White();
```

In the table below, indicate in the right-hand column the output produced by the statement in the left-hand column. If the statement produces more than one line of output, indicate the line breaks with slashes as in "a/b/c" to indicate three lines of output with "a" followed by "b" followed by "c". If the statement causes an error, fill in the right-hand column with either the phrase "compiler error" or "runtime error" to indicate when the error would be detected.

Statement	Output
`var1.method1();`	_____
`var2.method1();`	_____
`var3.method1();`	_____
`var4.method1();`	_____
`var1.method2();`	_____
`var2.method2();`	_____
`var3.method2();`	_____
`var4.method2();`	_____
`var1.method3();`	_____
`var2.method3();`	_____
`var3.method3();`	_____
`var4.method3();`	_____
`((Blue)var3).method1();`	_____
`((Red)var3).method2();`	_____
`((White)var3).method3();`	_____
`((White)var4).method3();`	_____
`((Green)var5).method3();`	_____
`((Red)var5).method1();`	_____
`((Blue)var6).method3();`	_____
`((Green)var6).method3();`	_____

4. Stacks/Queues, 10 points. Write a method called mirrorCollapse that takes a stack of integers as a parameter and that collapses the stack by combining pairs of values whose positions are a mirror image of each other. For example, suppose a variable s stores these values:

```
bottom [1, 2, 3, 40, 50, 60] top
        ^  ^  ^  ^   ^   ^
        |  |  +--+   |   |
        |  +---------+   |
        +----------------+
         mirror positions
```

The pairs of values in mirror image positions are the first and last (1 and 60), the second and fifth (2 and 50), and the third and fourth (3 and 40).

The method should add the first value to its mirror image, add the second value to its mirror image, add the third value to its mirror image, and so on. Thus, if we make the following call:

```
mirrorCollapse(s);
```

the stack should store the following values after the call:

```
bottom [43, 52, 61] top
```

If there is a value in the middle that has no mirror image, then it should not be altered. For example, if the stack had instead stored these values:

```
bottom [1, 2, 3, 4, 50, 60, 70] top
```

then it should store the following values after the method is called:

```
bottom [4, 53, 62, 71] top
```

Notice that the value 4 that was in the middle is unchanged. This example uses values with a regular pattern to make it easier to understand what is going on, but you should not assume anything about the sequence. For example, if s instead stored this sequence:

```
bottom [7, 1, 4, 18, 9, 23, 0, -5, 12] top
```

then after the method is called, it would store this sequence:

```
bottom [9, 41, 4, -4, 19] top
```

Your method should not change the stack if it has fewer than two values.

You are to use one queue as auxiliary storage to solve this problem. You may not use any other auxiliary data structures to solve this problem, although you can have as many simple variables as you like. You also may not solve the problem recursively. Your solution must run in O(n) time where n is the size of the stack. Use the Stack and Queue structures described in the cheat sheet and obey the restrictions described there (recall that you can't use the peek method or a foreach loop or iterator).

5. Collections Programming, 5 points. Write a method called recordTrip that records information about trips taken by people. Trip information will be stored in a map in which the keys are names of people and the values are sets of place names. The method will take as parameters the map followed by a person's name followed by a place name. For example, if we start with an empty map stored in a variable called trips and we make the following calls:

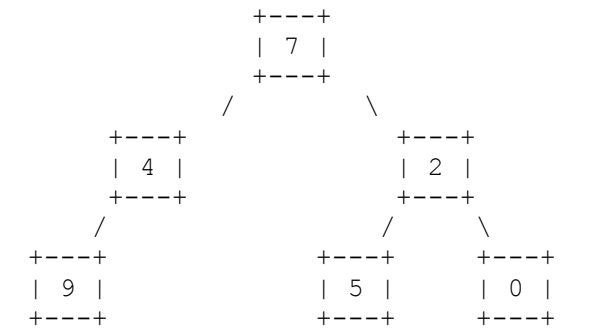
```
recordTrip(trips, "John", "London");
recordTrip(trips, "Sally", "Seattle");
recordTrip(trips, "John", "Paris");
recordTrip(trips, "Sally", "San Francisco");
recordTrip(trips, "John", "NYC");
recordTrip(trips, "John", "Paris");
```

the map would store the following values after these calls:

```
{John=[London, NYC, Paris], Sally=[San Francisco, Seattle]}
```

Notice that the map needs to construct a set for each person to store the names of the places they have visited. The sets it constructs should store the place names in alphabetical order. You may construct iterators and the sets that store place names, but you are not allowed to construct other structured objects (no string, set, list, etc.).

6. Binary Trees, 10 points. Write a method of the IntTree class called inorderList that returns a list containing the sequence of values obtained from an inorder traversal of the tree. For example, if a variable t stores a reference to the following tree:



then the call t.inorderList() should return the following list:

```
[9, 4, 7, 5, 2, 0]
```

Your method should construct an ArrayList to return. If the tree is empty, your method should return an empty list.

You are writing a public method for a binary tree class defined as follows:

```
public class IntTreeNode {
    public int data;          // data stored in this node
    public IntTreeNode left;  // reference to left subtree
    public IntTreeNode right; // reference to right subtree

    <constructors>
}

public class IntTree {
    private IntTreeNode overallRoot;

    <methods>
}
```

You are writing a method that will become part of the IntTree class. You may define private helper methods to solve this problem, but otherwise you may not call any other methods of the class. You may not construct any extra data structures other than the ArrayList you are returning.

7. Collections Programming, 10 points. Write a method called removePoints that takes a map and an index as parameters and that removes particular points from the map returning them in a set. The map this method will manipulate uses integer indexes as keys and store as values a list of points. For example, a variable called data might store the following:

```
{17=[[x=3,y=4], [x=12,y=6], [x=8,y=12], [x=3,y=6]],
 42=[[x=2,y=5], [x=3,y=3], [x=1,y=5], [x=4,y=2], [x=8,y=9]],
 308=[[x=1,y=2], [x=5,y=8], [x=4,y=4], [x=2,y=7], [x=3,y=9]]}
```

This map has three entries. The first entry associates the key 17 with a list of four points. The second associates the key 42 with a list of five points. The third associates 308 with a list that also has five points.

When the method is called, it will be passed the map and a key and it will return a set of points, as in:

```
Set<Point> result = removePoints(data, 42);
```

The method should manipulate the list of points for the given index, removing any points for which the x-value is less than the y-value and returning these points in a set. After the call above, result should be:

```
[[x=2,y=5], [x=1,y=5], [x=8,y=9]]
```

and data should store the following:

```
{17=[[x=3,y=4], [x=12,y=6], [x=8,y=12], [x=3,y=6]],
 42=[[x=3,y=3], [x=4,y=2]],
 308=[[x=1,y=2], [x=5,y=8], [x=4,y=4], [x=2,y=7], [x=3,y=9]]}
```

Notice that the index 42 is now associated with a list of just two points (the two that weren't removed). The method should return an empty set if there are no points to remove or if the index value has no corresponding entry in the map.

Your method should construct a set to return and may construct iterators, but you are not allowed to construct other structured objects (no string, set, list, etc.).

8. Comparable class, 10 points. Define a class called USCurrency that can be used to store a currency amount in dollars and cents (both integers) where one dollar is 100 cents. Your class should include the following methods:

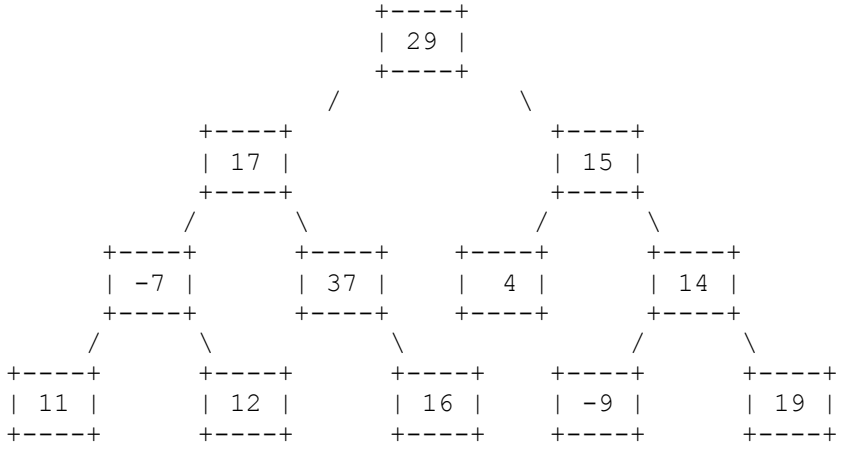
USCurrency(dollars, cents)	constructs a currency object with given dollars and cents
dollars()	returns the dollars
cents()	returns the cents
toString()	returns a String in standard $d.cc notation (-$d.cc for negative amounts)

A currency amount can be negative. The constructor should allow you to pass any values for dollars and cents, including negative values, and mixtures of negatives and positives. The cents method should return values in the range of 0 to 99 for nonnegative currency amounts and should return values in the range of 0 to -99 for negative currency amounts. For example, the constructor might be asked to work with 38 dollars and -413 cents, but that would be reported as 33 dollars and 87 cents.

Note that the toString method should return the amount in a standard format ($d.cc) with two digits for cents and with negative values indicated with a single minus sign in front of the dollar sign (-$d.cc). For example, 4 dollars and 5 cents would be expressed as "$4.05" while -19 dollars and -43 cents would be expressed as "-$19.43".

In addition, your class should implement the Comparable<E> interface. USCurrency objects should be compared in the obvious way, with smaller currency amounts considered "less" than larger currency amounts (e.g., -$13.45 < -$2.03 < $5.13 < $98.06).

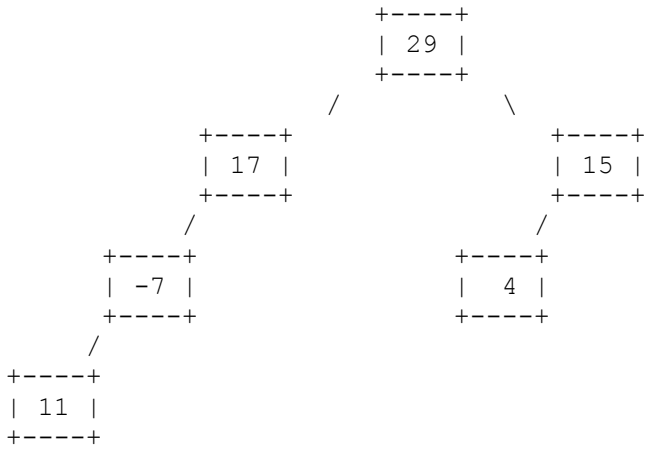
9. Binary Trees, 15 points. Write a method called limitPathSum that removes nodes from a binary tree of integers to guarantee that the sum of the values on any path from the root to a node does not exceed some maximum value. For example, suppose that a variable t stores a reference to the following tree:



Then the call:

```
t.limitPathSum(50);
```

will remove nodes so as to guarantee that no path from the root to a node has a sum that is greater than 50. This will require removing the node with 12 in it because the sum of the values from the root to that node is greater than 50 (29 + 17 + -7 + 12, which is 51). Similarly, we have to remove the node with 37 in it because its sum is too high (29 + 17 + 37, which is 83). Whenever you remove a node, you remove anything under it as well, so removing the node with 37 also removes the node with 16 in it. We also remove the node with 14 and everything under it because its sum is too high (29 + 15 + 14, which is 58). Thus, we end up with:



The method would be forced to remove all nodes if the data stored at the overall root is greater than the given maximum.

Assume that you are writing a public method for a binary tree class defined as follows:

```
public class IntTreeNode {
    public int data;          // data stored in this node
    public IntTreeNode left;  // reference to left subtree
    public IntTreeNode right; // reference to right subtree

    <constructors>
}

public class IntTree {
    private IntTreeNode overallRoot;

    <methods>
}
```

You are writing a method that will become part of the IntTree class. You may define private helper methods to solve this problem, but otherwise you may not call any other methods of the class. You may not construct any new nodes and you may not use any auxiliary data structure to solve this problem (no array, ArrayList, stack, queue, String, etc).

10. Linked Lists, 15 points.Write a method of the LinkedIntList class called switchPairsOfPairs that rearranges each successive sequence of 4 values by switching the order of the two pairs that make up the sequence. Suppose, for example, that a variable called list stores the following values:

```
[1, 2, 3, 4, 5, 6, 7, 8]
 |  |  |  |  |  |  |  |
 +--+  +--+  +--+  +--+
 pair  pair  pair  pair
```

As indicated, this list has four pairs. If the following call is made:

```
list.switchPairsOfPairs();
```

the following sequence would be produced:

```
[3, 4, 1, 2, 7, 8, 5, 6]
 |  |  |  |  |  |  |  |
 +--+  +--+  +--+  +--+
 pair  pair  pair  pair
```

Notice that the pair (1, 2) has been switched with the pair (3, 4) and that the pair (5, 6) has been switched with the pair (7, 8).

This example purposely used sequential integers to make the rearrangement clear, but you should not expect that the list will store sequential integers. It also might have extra values at the end that are not part of a group of four. Such values should not be moved. For example, if the list had stored this sequence of values:

```
[3, 8, 19, 42, 7, 26, 19, -8, 193, 204, 6, -4, 99, 2]
```

then a call on the method would have produced this sequence:

```
[19, 42, 3, 8, 19, -8, 7, 26, 6, -4, 193, 204, 99, 2]
```

Notice that the values 99 and 2 that appear at the end have not been moved because they are not part of a complete group of four values.

You are writing a public method for a linked list class defined as follows:

```
public class ListNode {
    public int data;       // data stored in this node
    public ListNode next;  // link to next node in the list

    <constructors>
}

public class LinkedIntList {
    private ListNode front;

    <methods>
}
```

Your method will become part of the LinkedIntList class. You may define private helper methods to solve this problem, but otherwise you may not assume that any particular methods are available. You are allowed to define your own variables of type ListNode, but you may not construct any new nodes, and you may not use any auxiliary data structure to solve this problem (no array, ArrayList, stack, queue, String, etc). You also may not change any data fields of the nodes. You MUST solve this problem by rearranging the links of the list.