CSE 373 Section 1: 143 Review

Stacks and Queues

(1) Write a method reverseHalf that reverses the order of half the elements of a queue of integers. Your method should reverse the order of all the elements in odd-numbered positions (position 1, 3, 5, etc) assuming that the first value in the queue has position 0. For example, if the queue originally stores this sequence of integers when the method is called:

front $[1, 8, 7, 2, 9, 18, 12, 0]$ back

Then the call of rearrange(q); should rearrange the queue to store the following sequence of values:

front $[1, 0, 7, 18, 9, 2, 12, 8]$ back

Notice that numbers in even positions (positions 0, 2, 4, 6) have not moved. That subsequence of integers is still: (1, 7, 9, 12). But notice that the numbers in odd positions (positions 1, 3, 5, 7) are now in reverse order relative to the original. In other words, the original subsequence: (8, 2, 18, 0) has become: (0, 18, 2, 8) You may use a single stack as auxiliary storage.

(2) Write a method switchPairs that takes a stack of integers as a parameter and that switches successive pairs of numbers starting at the bottom of the stack. For example, if the stack initially stores these values:

bottom $[3, 8, 17, 9, 99, 9, 17, 8, 3, 1, 2, 3, 4, 14]$ top

Your method should switch the first pair (3, 8), the second pair (17, 9), the third pair (99, 9), and so on, yielding this sequence:

bottom $[8, 3, 9, 17, 9, 99, 8, 17, 1, 3, 3, 2, 14, 4]$ top

If there are an odd number of values in the stack, the value at the top of the stack is not moved. Do not make assumptions about how many elements are in the stack. Use one queue as auxiliary storage.

(3) Write a method equals that takes as parameters two stacks of integers and returns true if the two stacks are equal and that returns false otherwise. To be considered equal, the two stacks would have to store the same sequence of integer values in the same order. Your method is to examine the two stacks but must return them to their original state before terminating. You may use one stack as auxiliary storage.

LinkedList

(1) Write a method reverse that reverses the order of the elements in the list. For example, if the variable list initially stores this sequence of integers:

$[1, 8, 19, 4, 17]$

It should store the following sequence of integers after reverse is called:

$[17, 4, 19, 8, 1]$

(2) Write a method removeEvens that removes the values in even-numbered indexes from a list, returning a new list containing those values in their original order. For example, if a variable list1 stores these values:

list1: $[8, 13, 17, 4, 9, 12, 98, 41, 7, 23, 0, 92]$

And the following call is made:

LinkedIntList list2 = list1.removeEvens();

After the call, list1 and list2 should store the following values:

list1: $[13, 4, 12, 41, 23, 92]$

list2: $[8, 17, 9, 98, 7, 0]$

Notice that the values stored in list2 are the values that were originally in even-valued positions (index 0, 2, 4, etc.) and that these values appear in the same order as in the original list. Also notice that the values left in list1 also appear in their original relative order. Recall that LinkedIntList has a zero-argument constructor that returns an empty list. You may not call any methods of the class other than the constructor to solve this problem. You are not allowed to create any new nodes or to change the values stored in data fields to solve this problem; You must solve it by rearranging the links of the list.

(3) Write a method doubleList that doubles the size of a list by appending a copy of the original sequence to the end of the list. For example, if a variable list stores this sequence of values:

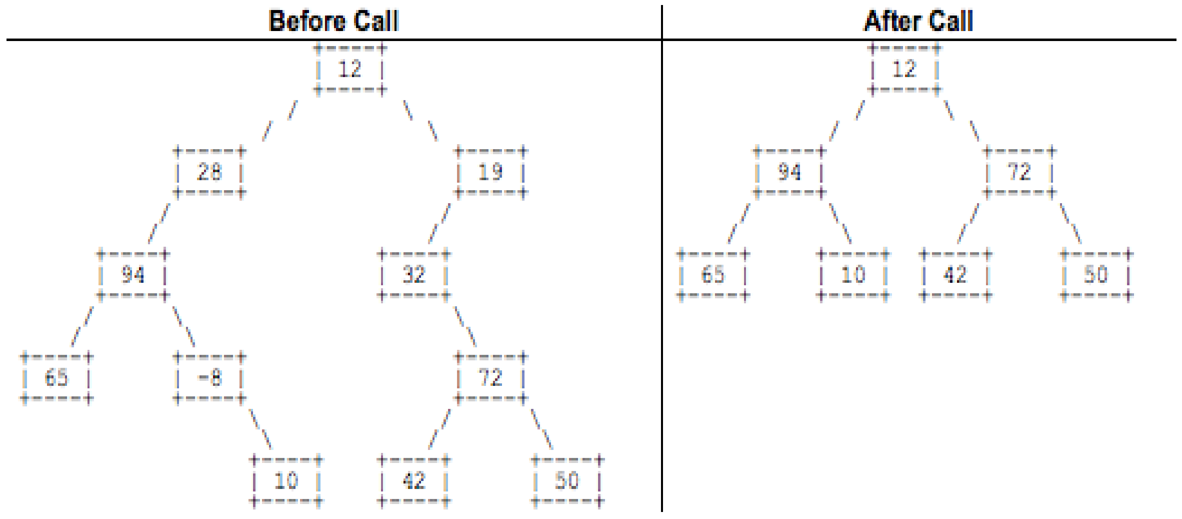
$[1, 3, 2, 7]$

And we make the call of list.doubleList(); then it should store the following values after the call:

$[1, 3, 2, 7, 1, 3, 2, 7]$

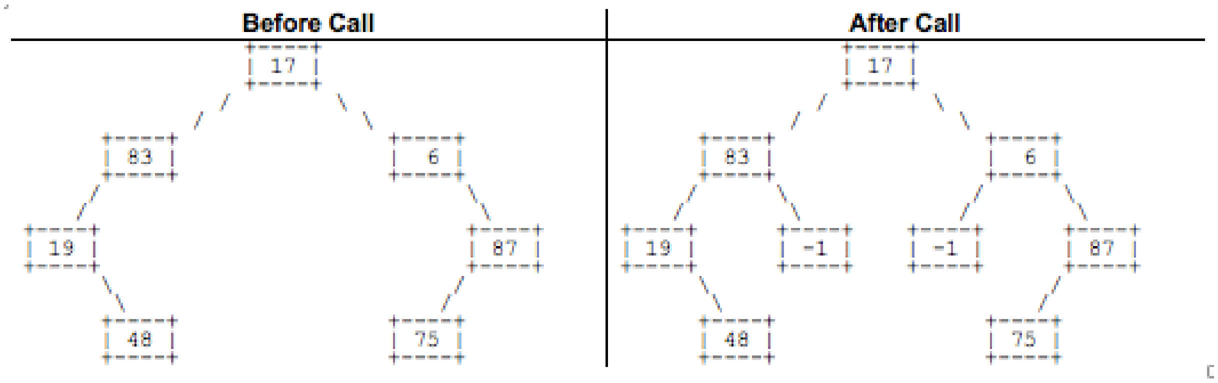
Binary Trees

(1) Write a method tighten that eliminates branch nodes that have only one child. For example, if a variable called t stores the tree below at left, the call of t.tighten(); should leave t storing the tree at right.



The nodes that stored the values 28, 19, 32, and -8 have been eliminated because each had one child. When a node is removed, it is replaced by its child. This can lead to multiple replacements because the child might itself be replaced (as in the case of 19 which is replaced by 32 which is replaced by 72).

(2) Write a method completeToLevel that accepts an integer n as a parameter and that adds nodes to a tree so that the first n levels are complete. A level is complete if every possible node at that level is non-null. We will use the convention that the overall root is at level 1, its children are at level 2, and so on. You should preserve any existing nodes in the tree. Any new nodes added to the tree should have -1 as their data. For example, if a variable called t refers to the tree below at right and you make the call of t.completeToLevel(3); , t should store the tree below at right after the call.



In this case, the request was to fill in nodes as necessary to ensure that the first 3 levels are complete. There were two nodes missing at level 3. Notice that level 4 of this tree is not complete because the call requested that nodes be filled in to level 3 only.

Keep in mind that your method might need to fill in several different levels. Your method should throw an IllegalArgumentException if passed a value for level that is less than 1.