QUEENS COLLEGE  Department of Computer Science
CSCI 313  Practice problems on Binary Trees

Instructor: Alex Ryba

These are some of the solutions to practice problems. Not all problems have solutions here.

Solutions to some older problems might not make use of generics. Generics are now required in this course.

**Problem 1** Consider the following Java program.

```
public class Problem2 {
  public static void main(String args[]) {
    BinaryTree<String> t = new BinaryTree<>();
    initializeTree(t);
    System.out.println(t.height());                                // line (a)
    for (Node<String> x:t.preOrder())
      System.out.print(x.getData() + " "); System.out.println();   // line (b)
    for (Node<String> x:t.postOrder())
      System.out.print(x.getData() + " "); System.out.println();   // line (c)
    for (Node<String> x:t.inOrder())
      System.out.print(x.getData() + " "); System.out.println();   // line (d)
  }
}
```

Assume that the class *BinaryTree* is a standard binarty tree as implemented in class and that the method *initializeTree* sets the tree to store the following data:

```
    A
  B   C
        D
       E F
```

(a) What is the output at line (a)?

3

(b) What is the output at line (b)?

A B C D E F

(c) What is the output at line (c)?

B E F D C A

(d) What is the output at line (d)?

B A C E D F

**Problem 2** A *parityOrder* traversal of a BinaryTree placess a node that has exactly one child after the subtree headed by its child. Otherwise the subtree headed by the left child comes first, followed by the node itself, and finally the subtree headed by the right.

Assume that *class BinaryTree* is implemented using the following skeleton Java code for *BNode* and *BinaryTree*.

```
public class BNode<T> {
  BNode<T> parent, left, right;
  T data;
  // constructors, getters and setters and other standard method are available
  // left and right are set to null in case corresponding children are absent
}

public class BinaryTree<T> {
   BNode<T> root;
   // standard BinaryTree methods are implemented
   public ArrayList<BNode<T>> parityOrder() {
      ArrayList<BNode<T>> answer = new ArrayList<BNode<T>>();
```

```
      parityOrder(root, answer);
      return answer;
   }
}
```

**Write a Java implementation for the auxiliary recursive method called parityOrder.**

**Answer:**

```
   public void parityOrder(BNode<T> p, ArrayList<BNode<T>> v) {
      if (p == null) return;
      if (p.getLeft() == null) {
         parityOrder(p.getRight(), v);
         v.add(p);
      }
      else {
         parityOrder(p.getLeft(), v);
         v.add(p);
         parityOrder(p.getRight(), v);
      }
   }
```

**Problem 3** The methods given below are used to calculate the value represented by an expression tree. Some pieces of code have been replaced by `PART (a)`, `PART (b)`, and so on. To answer the 5 parts of this question you should supply the `Java` code that was replaced. Each answer must fit on a single line.

In this problem, an expression is stored in a variable of type `BinaryTree<String>` whose nodes have type `BinaryNode<String>`. The `class BinaryTree` has a method `getRoot()` and the `class BinaryNode` has standard methods `getData()`, `getLeft()`, `getRight()`. If a main program declares a variable `BinaryTree<String> expression` and fills it with String data that represents the operations and numbers in an expression tree, then the instruction:

```
   System.out.println(evaluate(expression.root()));
```

prints the value of the expression.

The methods, with parts omitted are as follows:

```
// Method to find the value represented by an expression tree
  static double evaluate(BinaryNode<String> root) {
    if (!isOperation( PART(a) ))
       return PART(b);
    return apply( PART(c) , PART(d) , PART(e));
  }


// Method to apply an operation to two numbers
  static Double apply(String op, double x, double y) {
    switch (op) {
    case "+": return x + y; case "-": return x - y; case "*": return x * y; case "/": return x / y;
    }
    return null;
  }

// Method to decide whether a node stores an operation
  static boolean isOperation(BinaryNode<String> n) {
    String x = n.getData();
    if (x.equals("+") || x.equals("-") || x.equals("*") || x.equals("/")) return true;
    return false;
  }
```

(a) Give a replacement for PART (a)

**Answer:** `root`

(b) Give a replacement for PART (b)

**Answer:** `Double.parseDouble(root.getData())`

(c) Give a replacement for PART (c)

**Answer:** `root.getData()`

(d) Give a replacement for PART (d)

**Answer:** `evaluate(root.getLeft())`

(e) Give a replacement for PART (e)

**Answer:** `evaluate(root.getRight())`

**Problem 4** The *class BNode* implements a generic binary tree node. The class provides the following instance variables and methods.

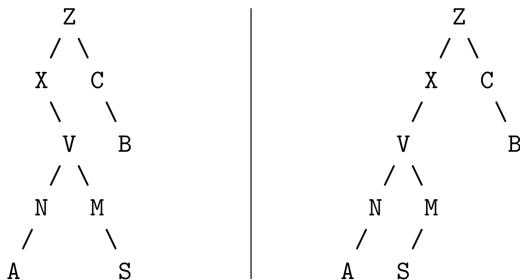
```
public class BNode<T> {
   BNode<T> parent, left, right;
   T data;
   public BNode(T data, BNode<T> parent, BNode<T> left, BNode<T> right) // code omitted
   // getters and setters for all instance variables, code omitted

   public void leftShift()             // to be written

}
```

Write complete code for the method *leftShift* that acts on the subtree of descendants of the `BNode`. For any node `lower` in this subtree that has only a right child, that right child should be moved to become a left child.

For example, if `node.leftShift()` is called at the node storing **X** in the tree of the left hand diagram, the result is the tree in the right hand diagram.




Code longer than 15 lines may be subject to a penalty. Code with a running time worse than $O(n)$ (where $n$ is the number of nodes in the subtree being adjusted) will be subject to a penalty.

**Answer:**

```
   public void leftShift() {
       if (left == null && right == null) return;  // base case
       if (left == null) {
           left = right;
           right = null;
       }
       left.leftShift();
       if (right != null) right.leftShift();
   }
```