QUEENS COLLEGE — Department of Computer Science
CSCI 313 — Practice problems on General Trees and 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 A new method *miniRemove* in a *GeneralTree* is applied to a node of the tree as follows. If the node is a leaf, it is removed. Otherwise its data value is replaced by the data value of its first child and its firstChild is recursively removed by the method miniRemove.

For example if the GeneralTree is as follows:

```
  1
    2     3  4
      5 6      7
                 8
```

then if miniRemove is applied to the root, the tree changes to:

```
  2
    5   3  4
      6      7
               8
```

Assume that GeneralTrees are implemented as in class, based on the following skeleton Java code for the classes GNode and GeneralTree.

```
public class GNode<T> extends Node<T> {
  //  The inherited instance variables from class Node<T> are:  parent and data
   ArrayList<GNode<T>> children;      //   can be empty, but is never null
  //   standard methods and constructors omitted.
}

class GeneralTree<T> extends Tree<T> {
  //  The inherited instance variable from class Tree<T> is:    Node<T> root
  //  standard methods and constructors omitted
  //  The new method has title:
  public void miniRemove(GNode<T> n) {
     // code omitted
  }
}
```

Write a Java implementation for the method called miniRemove.

Hint: You might find it useful to call standard *Tree* methods such as *isRoot*, *isLeaf* and the standard *GeneralTree* method *remove*. The *GNode* method *get(i)* that returns the i^{th} child of a node is also useful here.

You may assume that the parameter is a node of the tree, in particular it is not null.

Answer:

```
public void miniRemove(GNode<T> n) {
   if (isLeaf(n)) remove(n);
   else {
     GNode<T> c = n.get(0);
     n.setData(c.getData());
     miniRemove(c);
   }
}
```

Problem 2 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 3 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);
   }
}
```