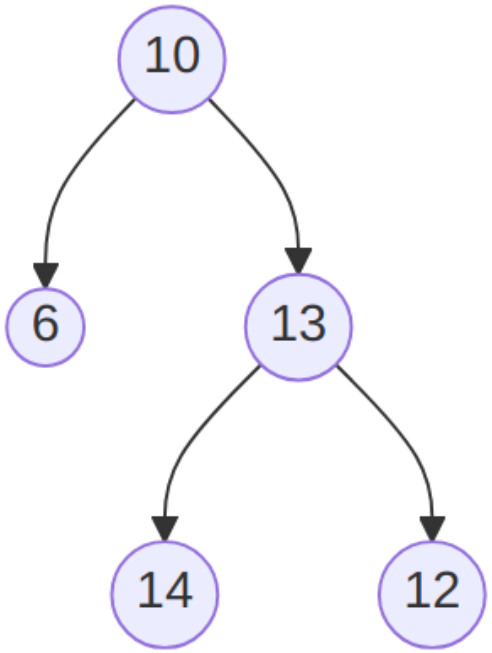
Instructions: This exam is to be taken in silence, without notes, books, or electronic devices (including "smart" watches or earbuds). The time limit to complete it is the duration of the class period (1 hour and 15 minutes). Answer the following questions and problems, trying to be as clear and as accurate as possible. Take your time to read the statements carefully before trying to answer them. If you need more space, write on the back of your test page and indicate it clearly. When writing code, make sure your special punctuation characters are legible, and your lowercase and uppercase letters are easy to distinguish. As usual, every statement or series of statement is assumed to be in a valid class and method, and you can use the `C.W( )` and `C.WL( )` abbreviations.

____ / 15 pts. **Problem 1** Consider the following tree:



1. Explain why it is **not** a binary search tree.

2. Pick one among *inorder*, *preorder* and *postorder* traversal, and give
   (a) A brief description of how it proceeds,

   (b) What it would produce for the given tree.

____ / **30 pts.** **Problem 2** Consider the implementation of "random" binary trees shared on page 6, and note that the `Insert(T dataP, Node nodeP)` method uses the `gen.NextDouble() > 0.5` test that will be randomly **`true`** half of the time, and **`false`** the other half.

1. Explain the `T dataP = default(T)` part of the `Node` constructor.

2. Write a `ToString` method for the `Node` class, remembering that only a node `Data` needs to be part of the `string` returned.

3. Write a series of statements that would
   (a) create a `RBTree` object,
   (b) insert the values 1, 2, 3, and 4 in it (in this order).

4. Make a drawing of a possible `RBTree` obtained by executing your code.

5. Write a `Find` method that takes one argument `dataP` of type `T` and returns **`true`** if `dataP` is in the `RBtree` calling object, **`false`** otherwise.

____ / **20 pts.** **Problem 3** Consider the "usual" implementation of lists:

```
public class CList<T>
{
    private class Cell
    {
        public T Data { get; set; }
        public Cell Next { get; set; }
        public Cell(T dataP, Cell nextP)
        {
            Data = dataP;
            Next = nextP;
        }
    private Cell first;
    public CList(){first = null;}
}
```

1. Write a `AddF` method that takes an argument of type `T` and adds it "to the left" of the `CList` calling object.

2. Write a `RemoveL` method that remove the value "to the right" of the `CList` calling object and returns it.

3. If those were the only two methods to add to and to remove from the list, what would be the name of the data-structure we actually just implemented?

____ / **20 pts.** **Problem 4** Answer the following short questions (checking all that applies):

1. A queue is generally endowed with operations called…
   ☐ Requeue ☐ Dequeue ☐ Enqueue ☐ Unqueue
2. A stack is generally endowed with operations called…
   ☐ Pop ☐ Push ☐ Pull ☐ Peek ☐ Pounce
3. LIFO stands for…
   ☐ Least Is First Out ☐ Last Is First Outside ☐ Last In First Out ☐ Low Input Fast Output
4. A queue implements which principle?
   ☐ LIFO ☐ FIFO ☐ LILO ☐ FOFI
5. Implementing a list as a doubly linked list (as opposed to singly linked list) allows to …
   ☐ use fewer attributes. ☐ keep track of the end of the list.
   ☐ store more elements. ☐ insert at the beginning of the list faster.
6. A binary search tree has…
   ☐ exactly one root. ☐ the ability to store the same value multiple times.
   ☐ no leaves. ☐ ways of finding a value faster than linear time in its size.

____ / **10 pts.** **Problem 5** Suppose given an empty `Queue` object, and assume that we store the values 10 and 20 (in that order), and then remove one and insert 30. Draw the resulting queue, labelling explicitly the front (or beginning) and end of your queue.

____ / **10 pts.** **Problem 6** Suppose given an empty `Stack` object, and assume that we store the values 10 and 20 (in that order), and then remove one and insert 30. Draw the resulting stack, labelling explicitly the bottom and top of your stack.

```
public class RBTree<T>
    {

    private class Node
        {
        public T Data { get; set; }
        public Node left;
        public Node right;
        public Node(
            T dataP = default(T),
            Node leftP = null,
            Node rightP = null
            )
            {
                Data = dataP;
                left = leftP;
                right = rightP;
            }
        }

    private Node root;

    public RBTree( )
        {
            root = null;
        }

    public void Insert(T dataP)
        {
            root = Insert(dataP, root);
        }

    private Node Insert(T dataP, Node nodeP)
        {
            if (nodeP == null)
            {
                return new Node(dataP, null, null);
            }
            else
            {
                Random gen = new Random( );
                if(gen.NextDouble( ) > 0.5)
                {
                    nodeP.left = Insert(dataP, nodeP.left);
                }
                else
                {
                    nodeP.right = Insert(dataP, nodeP.right);
                }
            }
            return nodeP;
        }
    }
```