Name: _____

Please follow the submission procedure used for the previous assignment. Reminder: You are welcome to use any inanimate resources (e.g., books, Web sites, publicly available code) to help you with your work. However, *all such help must be clearly noted* in your submissions. Further, no matter what you use, *you must be able to explain, in detail, how it works.* (You may be called upon to explain your homework in person.) Refer to the class policy for details, and ask for clarifications if you are unsure if something is allowed. Questions marked with a $\star$ are optional but you are encouraged to answer them for extra credit.

Important: Make sure you follow the packaging and submission procedure carefully. Include a README file with information on the rest of the files in your submission, and instructions on how to test your work. Also include a Makefile that enables the make program to compile your source. Your submission must contain *only* source code. Test unpacking, compiling, and running your packaged files on gandalf prior to your final submission.

1. (1 pt.) Write your name in the space provided above.

2. (1 pt.) Package and submit your solutions to the programming questions. *After* uploading your jar file to the FTP server, complete the following:
   File name: _____ Size, in bytes: _____

3. (13 pts.) In class, we defined the function iot as

$$\text{iot}(t) = \begin{cases} \text{iot}(l) \cdot (n) \cdot \text{iot}(r) & \text{if } t = (n, l, r) \\ () & \text{otherwise} \end{cases}$$

   where () denotes the empty sequence, (n) denotes the sequence with a single element n, and $\cdot$ is the concatenation operator on sequences.

   (a) Trace the evaluation of $\text{iot}(t_0)$ where t_0 is the tree defined in Question 8 of Homework 1. Clearly indicate the arguments and result of each invocation of the iot function.

   (b) Prove that this function results in an *inorder* traversal[1] of a binary tree. In particular, prove that if t is a binary search tree then $\text{iot}(t)$ is the sequence of the keys of t in sorted order.

[1] Mark Allen Weiss, *Data Structures and Problem Solving Using Java*, 3rd edition (Addison-Wesley, 2006), §18.4, p. 611.

[additional space for answering the earlier question]

[additional space for answering the earlier question]

4. (15 pts.) Provide suitable recursive definitions of functions preord(t) and postord(t) corresponding to the *preorder* and *postorder* traversals of a binary tree t, using a notation similar to that in Question 3. Prove that each function produces a sequence of the keys in t in the appropriate order.

[additional space for answering the earlier question]

5. (30 pts.) A rooted, *ordered*, labeled tree is a rooted, labeled tree (as defined in the textbook[2]) in which there is a well defined order on the children of each node. For example, the tree depicted in Figure 18.1 may be interpreted as a rooted, ordered, labeled tree by imposing a left-to-right order on the children of each node. Thus, D is the third child of A, and G is the second child of B.

We may represent such trees in a two-dimensional plain-text format by using indentation to suggest parent-child relationships. For example, the tree of Figure 18.1 in the textbook may be represented as indicated on the right where, for clarity, we use the ␣ symbol to denote a single space character. In more detail, the *two-dimensional text representation* of a tree t contains one line for each node of t. The line representing a node $n \in t$ consists of n's label prefixed with $3d$ spaces, where d is the depth of n in t. Further, all the lines representing descendants of a node precede the line representing that node's right sibling, if any. In our example, all nodes in B's subtree are represented before B's right sibling C.

```
A
␣␣␣B
␣␣␣␣␣␣␣F
␣␣␣␣␣␣␣G
␣␣␣C
␣␣␣D
␣␣␣␣␣␣␣H
␣␣␣E
␣␣␣␣␣␣␣I
␣␣␣␣␣␣␣J
␣␣␣␣␣␣␣␣␣␣K
```

(a) Provide a precise and compact recursive definition of a function tdtr that maps trees to their two-dimensional text representations, using the definition of iot in Question 3 as a guide. You may use the operator $\circ$ for text concatenation and the symbols ␣ and $\hookleftarrow$ to represent the space and newline characters. [Hint: Use an auxiliary function $a(d, s)$ that is invoked on subtrees s of the input tree t, with d equal to the depth the root of subtree s in the tree t; initially, $\text{tdtr}(t) = a(0, t)$.]

(b) Characterize, as precisely as possible, the running time of a direct implementation of your recursive definition, as a function of the size of the input tree.

(c) Trace the evaluation of $\text{tdtr}(t_1)$ using a direct implementation of your recursive definition, where t_1 is the tree from Figure 18.1 in the textbook. For each recursive function invocation, indicate the function arguments and result.

[2] *Idem*, p. 596.

[additional space for answering the earlier question]

[additional space for answering the earlier question]

6. (40 pts.) We define a *simple digital trie*[3] to be a nonempty rooted labeled tree[4] in which each non-root node is labeled with a single digit $(0, \ldots, 9)$, and in which no two siblings have identical labels. The root r has an empty label. We associate a trie node $n(z)$ with every non-negative integer z as follows: If $z < 10$ then $n(z)$ has label z and parent r. Otherwise $n(z)$ has label $z \bmod 10$ and parent $n(\lfloor z/10 \rfloor)$. A *marked simple digital trie* is a simple digital trie in which each node may be associated with an optional *mark* (separate from its label). A marked simple digital trie is said to *contain a key* (non-negative integer) z if it contains a *marked* node $n(z)$ (and the other nodes implied by the recursive definition of $n(z)$, either marked or unmarked). A marked simple digital trie is said to *represent a set K of keys* if it contains exactly the keys in K (i.e., all the keys in K and no others).

   (a) Depict a marked simple digital trie that represents the following set of keys, using the usual graphical notation for labeled trees, and using the character $*$ to adorn marked nodes: $\{1, 3, 343, 2939, 48902, 22, 983001, 344, 35, 129\}$.

   (b) Describe simple algorithms for the following, by providing pseudocode or very precise English descriptions.

      i. to determine whether a marked simple digital trie T contains a key k.

      ii. to remove a key k from a marked simple digital trie T. [Hint: Check that the algorithm removes *only* k.]

   (c) Characterize, as precisely as possible, the running time of the algorithms you describe for Question 6b as a function of the inputs T and k.

   (d) Prove or disprove: If T_1 and T_2 are two marked simple digital tries representing a set K of keys, then T_1 is isomorphic to T_2.

[3] usually pronounced "try," although some prefer "tree."

[4] Weiss, *op. cit.*, p. 596.

[additional space for answering the earlier question]

[additional space for answering the earlier question]

7. (20 pts.) ⋆ Provide a suitable recursive definition of a function levelord(t) corresponding to the *level-order* traversal[5] of a binary tree t, using a notation similar to that in Question 3. Prove that your function produces a sequence of the keys in t in the level order.

[5] *Idem*, §18.4.4, p. 622.

8. (100 pts.) Implement a simple record manager using the marked simple digital trie of Question 6, as detailed below. The record manager stores *key-data pairs* of the form (k, d) where k is a non-negative integer and d is floating point number. (You may assume that k and d fit in Java's `int` and `float` types.) All user interaction with the record manager is through a text-mode command language based on the standard-input/standard-output interface. The input consists of one command per line. The record manager reads and responds to each command in turn. Except for the `xp` command, the response to each command is also a single newline-terminated line. The syntax and semantics of the commands, and the desired outputs, are as follows. As before, we use the symbol ␣ to denote a single space character. The commands `xs`, `xh`, `xa`, and `xb` use the definitions in the `Tree` interface of Question 11 in Homework 1.

command	actions
`s`␣`k`␣`d`	The pair (k, d) is stored in the record manager. If a pair of the form (k, d') already exists, then the new pair replaces it. The output is an empty line (single newline character).
`e`␣`k`	If the record manager contains the key k then the output is `1` else the output is `0`.
`r`␣`k`	The data d associated with key k is retrieved from the record manager. The output is d. If the key k is absent from the record store, the output consists of an empty line.
`xs`	The output is the size of the trie (a single integer).
`xh`	The output is the height of the trie.
`xa`	The output is a list of the labels of the trie nodes in preorder, with labels of marked nodes suffixed with a single `*` character. These labels (some with suffixes) are to be printed on a single line, with a single space separating adjacent entries, with no additional punctuation such as commas or brackets.
`xb`	The output is similar to that of `xa` but the nodes are to be listed in postorder.
`xp`	The output consists of the two-dimensional text representation (Question 5) of the current state of the marked simple digital trie used to store the records, with the labels of marked nodes suffixed with a single `*` character.

Ensure that (1) your program produces exactly the output described above, with no spurious text such as extra spaces or newlines, prompts, or other messages and (2) your program reads from standard input and writes to standard output, with no additional assumptions on either.

9. (20 pts) ⋆ A *shortened digital trie* is similar to a simple digital trie but nodes that have no siblings are merged into their parents, with a concatenation of labels. That is, if a node n_1 with label d_1 has a single child n_2 with label d_2, then n_2 is merged into n_1, and the label of n_1 is changed to $d_1 \cdot d_2$.

   (a) Provide an appropriate definition for a *marked shortened digital trie*, by analogy with the marked simple digital trie.

   (b) Repeat Question 6, replacing "simple digital trie" with "shortened digital trie" throughout.

   (c) Repeat Question 8 using shortened digital tries instead of simple digital tries. Submit your implementation with the rest of your work, noting all necessary details in your README file.