**CS-3114 Spring 2014 – Project 4: File-based BST**

**Due:** May 2, 9pm.
Late Policy as in the syllabus: 10% per day, maximum of 3 days.

**Assignment:**

For this project, you will implement a persistent database index that is stored on disk. While B-Trees are commonly used for this purpose, we will use the simpler BST. The index is in the form of a Binary Search Tree that is stored as a binary file instead of in memory (e.g. OpenDSA 9.5). The BST nodes are stored as sequential records in a binary file. The file will also maintain a FreeList of deleted records that can be re-used. The index will store records that contain a floating point key and an integer data value, with no duplicate keys.

**Design & Implementation Requirements:**

As usual, you are expected to use good object-oriented design and documentation in your solution. You are not required to make your BST generic. It is important to follow all of the rules to ensure that your solutions match the WebCAT solution.

***Binary Search Tree:***

The BST enables efficient access to the data by key. The BST must be stored only in the binary file. You are **NOT allowed** to read in and store the tree structure in memory. Only individual nodes may be stored in memory temporarily as local variables during tree processing. You may store the root byte position in memory.

During node insertion, the new BST record should first try to pop a record from the FreeList for reuse. If the FreeList is empty, then a new record should be appended to the database file for the new node.

During node deletion, use the "promote the minimum value of the right subtree" rule as in Project1. When promoting the minimum node, do **NOT move** its record in the file (e.g. do not copy its value to the record previously occupied by the deleted node) -- once a record is written to file, its value should **never** be moved to another record. The deleted record should then be pushed onto the FreeList.

***FreeList:***

The goal of the FreeList is to keep track of records that have been deleted in the file and whose space can therefore be reused by insertions (e.g. OpenDSA 5.6). The FreeList is represented as a LIFO Stack of available records. The FreeList is maintained without adding additional space to the file by simply reusing the empty record space to store the linked list pointers (e.g. OpenDSA 11.3). The FreeList must be stored only in the binary file. You are **NOT allowed** to read in and store the FreeList structure in memory. Only individual FreeList nodes may be stored in memory temporarily as local variables during list processing. You may store the head byte position in memory. FreeList operations must run in O(1) time.

When a deleted tree record is added to the FreeList, reuse its first 8 bytes to store the pointer to the next FreeList node, and reset the remaining bytes to 0 for easier debugging.

***File Structure:***

The database file is structured as follows. This is a **binary file, not a text file**.

- The first **8 bytes** stores the BST root pointer as a "long". It contains the byte position (offset) in the file of the first byte of the BST root node. Root pointer value 0 means an empty BST.
- The second **8 bytes** stores the FreeList head pointer as a "long". It contains the byte position in the file of the first byte of the FreeList head node. Head pointer value 0 means an empty FreeList.
- The rest of the file is a sequence of records of size **24 bytes** each, with no extra space between them. Each record is either a BST node or a FreeList node.
    - BST node records are as follows:
        - **4 byte** "float" key value (no duplicates allowed),
        - **4 byte** "int" data value,
        - **8 byte** "long" left child pointer, as a byte position in file (0 = null),
        - **8 byte** "long" right child pointer, as a byte position in file (0 = null).
    - FreeList node records are as follows:
        - **8 byte** "long" next node pointer, as a byte position in file (0 = null).
        - the remaining **16 bytes** are unused and must be reset to all 0's.

Thus, the file must be 16+24*(B+F) bytes in size, where B is the number of BST nodes and F is the number of FreeList nodes.

**Input and Output:**

All input commands should be read from System.in, and any output should be directed to System.out. Blank input lines are ignored, and lines beginning with "#" character are ignored as comments. See the sample output for detailed syntax.

Your database is invoked as follows (this implies a class FileBST containing "static void main(String[] args)" in the default package):

```
$ java FileBST filename
```

Output: **database new** filename
or: **database existing** filename

The command line argument, in args[0], is the name of the file (within the current working directory) to use for storing the database BST index. This file may or may not exist prior to invocation. If it does not, create it and start with an empty database. If it does, you may assume it contains a valid database as specified above, and should use it as the initial database. After program completion, the database file should contain the valid database as a result of processing the following commands.

Input: **insert** key value
Output: **insert** key value **at** bytePosition
or: **insert** key value **duplicate**

Input: **find** key
Output: **find** key value **at** bytePosition
or: **find** key **not found**

Input: **delete** key
Output: **delete** key value **at** bytePosition
or: **delete** key **not found**

Input: **tree**
Output: **tree empty**
or: **tree**
>key value **at** bytePosition
Where the BST records are printed sorted by the key in ascending order, each record on a separate line, and the number of ">" characters at the beginning of the line is the depth of the node in the tree (with root at depth 0).

Input: **freelist**
Output: **freelist empty**
or: **freelist**
i **at** bytePosition
Where the FreeList records are printed in LIFO stack order, each record on a separate line, and the number "i" at the beginning of the line is the numbered order of records starting at 1.

**Deliverables:**

You will submit your project code through the automated Web-CAT server. Links to the Web-CAT client are posted at the class website. If you make multiple submissions, only your last submission will be evaluated. There is no limit to the number of submissions that you may make.

You are required to submit your own test cases with your program, and part of your grade will be determined by how well your test cases test your program, as defined by Web-CAT's evaluation of code coverage. Of course, your program must pass your own test cases. Part of your grade will also be determined by test cases that are provided by the graders. Web-CAT will report to you which test files have passed correctly, and which have not. Note that you will not be given a copy of grader's test files, only a brief description of what each accomplished in order to guide your own testing process in case you did not pass one of our tests.

If submitting through Eclipse, the format of the submitted archive will be managed for you. If you choose not to develop in Eclipse, you will submit either a ZIP-compressed archive (compatible with Windows ZIP tools or the Unix zip command) or else a tar'ed and gzip'ed archive. Either way, your archive should contain the source code for the project only (no .class files or binary resources).

**Honor Code:**

You may only use code you have written, either specifically for this project or for earlier programs, or code taken from the textbook. Note that the textbook code is not designed for the specific purpose of this assignment, and is therefore likely to require modification. It may, however, provide a useful starting point. It is acceptable to share input and output files on the class forum, but not code.

Your project submission must include a statement, pledging your conformance to the Honor Code requirements for this course. Specifically, you must include the following pledge statement near the beginning of the file containing the function main() in your program. Programs that do not contain this pledge will not be graded.

```
// On my honor:
//
// - I have not used source code obtained from another student,
//   or any other unauthorized source, either modified or
//   unmodified.
//
// - All source code and documentation used in my program is
//   either my original work, or was derived by me from the
//   source code published in the textbook for this course.
//
// - I have not discussed coding details about this project with
//   anyone other than the instructor, ACM/UPE tutors or the TAs assigned
//   to this course. I understand that I may discuss the concepts
//   of this program with other students, and that another student
//   may help me debug my program so long as neither of us writes
//   anything during the discussion or modifies any computer file
//   during the discussion. I have violated neither the spirit nor
//   letter of this restriction.
```