$Id: asg3j-airport-bstree.mm,v 1.8 2016-02-09 13:16:58-08 - - $
PWD: /afs/cats.ucsc.edu/courses/cmps012b-wm/Assignments/asg3j-airport-bstree
URL: http://www2.ucsc.edu/courses/cmps012b-wm/:/Assignments/asg3j-airport-bstree/

## 1. Overview

In this assignment you will implement a binary search tree data structure that will support airport code queries. Your program will not attempt to create a balanced tree, since that is beyond the scope of this course, but will rather depend on input being suitably ordered.

## 2. Program specification

The program specification is presented in the form of a Unix `man`(1) page.

**NAME**

airport — look up airport name given airport code

**SYNOPSIS**

**`airport`** [**`-d`**] *database*

**DESCRIPTION**

An airport database is read in from the filename specified on the command line. Then the standard input is read, expecting one airport code per line. For each code read, the name of the airport is printed.

**OPTIONS**

**`-d`** If the **`-d`** option is specified, then the airport database is displayed in debug format and the standard input is ignored.

**OPERANDS**

Exactly one operand, the airport database, is required. Each line of the input database consists of an airport code, followed by a colon, followed by the name of the airport. Any line consisting only of white space or whose first non-white-space character is a hash (**`#`**) is ignored as a comment. Airport codes in the database are always stored in upper case. If an input airport code is in lower case, it is converted to upper case. The standard input consists of one airport code per line. Empty lines and lines beginning with a hash are ignored. Lookups of the airport codes are case insensitive. Thus, for example, **`sjc`** will be looked up as **`SJC`**.

**EXIT STATUS**

0 No errors were detected.

1 The database could not be opened or any lines from it were invalid.

## 3. Implementation strategy

As usual, you need to implement your program one step at a time. Following is a suggested implementation sequence :

(1) Begin with the main program. Handle options analysis, and for the database, read and print each line as it is seen. Print out each database line as two separate fields.

(2) Look for and fix any remaining bugs in the main program. Print a usage message if the arguments to the program are not correct.

(3) Make use of the sample class `treemap` in order to implement your insertion and lookup. Before submitting your program, you must remove all references to `java.util.TreeMap` from your code. The use of the standard library's `TreeMap` is just for you to implement your main program.

(4) Remove `TreeMap` from `treemap.java` and implement `get` and `put` properly.

(5) For `get`, perform a binary search on the tree. Return `null` for not found.

(6) For `put`, perform a binary search to find the tree node containing the key. If found, return the previous value, and replace the new value in the tree.

(7) If not found, create a new node as a new leaf node and insert the key and value fields. You are not expected to balance the tree. The test data has been cooked to produce a reasonably balanced tree, but balancing algorithms are beyond the scope of this course.
[`http://en.wikipedia.org/wiki/Red-black_tree`]

(8) Add an inorder traversal function which prints the contents of each tree node. For each node found in the tree, print its depth, key, value, left child, and right child.

(9) If the `-d` option is specified, then `stdin` is ignored and the database is printed in debug format, using a depth-first in-order traversal.

(10) The depth is the distance of a node from the root, with the root having depth 0. When printing the left and right child pointers, the default `toString` method will be used, which will print out the node's identity. Use the format

```
"%3d \"%s\" \"%s\" %s %s%n"
```

which will produce output like

```
  3 "sjc" "San Jose" treemap$tree@1fdc96c treemap$tree@b2fd8f
```

## 4. What to submit

Submit the files `README`, `Makefile`, and all necessary Java source files. Make sure all of the necessary class files are put in the jar file. Use `grep TreeMap *java` to verify that you have eliminated all references to `TreeMap`. Note that the name of the class you are writing is `treemap`, spelled in lower case. If you are doing pair programming, also submit the `PARTNER` file.