Project 4 – AVL Trees

Computer Science 2413 – Data Structures
Fall 2015

This project is individual work. Each student must complete this assignment independently.

User Request:

"Create a simple system to read, efficiently merge, efficiently purge, sort, search, and write exoplanetary data, providing fast lookup by name."

Objectives:

1.	Use C++ file IO to read and write files, while using C++ standard I/O (`cin` and `cout`) for user interaction, using appropriate exception handling and giving appropriate error messages.	*5 points*
2.	Encapsulate primitive arrays inside a templated class that provides controlled access to the array data, retains information on array capacity and use, and can be used to store data of any class or primitive type. Integrate appropriate exception handling into the templated array class.	*10 bonus points** (Project 1)
3.	Efficiently sort and search the data based on the field specified by the user. Integrate appropriate exception handling into classes that implement searching and sorting.	*15 bonus points** (Project 1)
4.	Encapsulate linked lists inside a templated class that provides controlled access to the linked-list data, retains information on linked list size, and can be used to store data of any class or primitive type. Integrate appropriate exception handling into the templated linked list class.	*25 bonus points** (Project 2)
5.	Encapsulate linked hash tables inside a templated class that provides controlled access to the linked hash table data, retains information on linked hash table capacity and load factor, and can be used to store data of any class or primitive type. Integrate appropriate exception handling into the templated linked hash table class.	*25 bonus points** (Project 3)
6.	Encapsulate AVL trees inside a templated class that provides controlled access to the AVL tree data, retains information on AVL tree size (number of entries), and can be used to store data of any class or primitive type.	*10 points*
7.	Store exosystems in an AVL tree sorted by name.	*20 points*
8.	Find exosystems stored in the AVL tree based on name.	*5 points*
9.	Remove exosystems from the AVL tree based on name.	*20 points*
10.	Provide an in-order traversal of the AVL tree.	*5 points*
11.	Integrate appropriate exception handling into the templated AVL tree class.	*5 points*
►	Develop and use an appropriate design.	*15 points*
►	Use proper documentation and formatting.	*15 points*

*Note that bonus points listed here apply to particular projects and cannot cause your score on that project to exceed 100.

Description:

For this project, you will revise and improve ExoPlanIt3.0 from Project 3 in one important way. You are encouraged to reuse and build on your code from Project 3. ExoPlanIt4.0 will have the same basic functionality as ExoPlanIt3.0 but it will have one major change "under the hood"– because it was very time-inefficient to keep the list of exosystems in a linked list that was always sorted by name while data was read in, merged, and purged, ExoPlanIt4.0 will instead keep an AVL tree of exosystems using their names as keys. This will allow for exosystems to be inserted and removed from in $\Theta(log_2 n)$ time rather than $\Theta(n)$ time (where n is the number of exosystems in the data).

Note that ExoPlanIt4.0 will still store the lists of exoplanets within each system using linked lists since there are very few exoplanets within each system, so the advantages of an AVL tree over a linked list are minimal. Also, ExoPlanIt4.0 will still copy the exoplanets to a linked hash map for fast lookup by name and will still store the list of exoplanets to be sorted and searched using fields besides name using a resizable array.

Operational Issues:

From a user interface perspective, ExoPlanIt4.0 will behave as described for ExoPlanIt3.0, except that the merge and purge operations may take noticeably less time.

Implementation Issues:

In most areas, ExoPlanIt4.0 will be implemented just as was ExoPlanIt3.0. This includes how ExoPlanIt reads files and prints data, carries out user interaction via standard in and standard out, encapsulates C primitive arrays, how exception handling is implemented for arrays and similar classes, and how the lists of exoplanets within each system are stored as linked lists. The big implementation change will be the data structure used to store the exosystems based on names. For ExoPlanIt4.0, you are no longer allowed to store this data in a linked list, you need to use an AVL tree instead. Similarly, display/printing option O should now be accomplished by performing an in-order traversal of the AVL tree of systems and performing a linear traversal within each system's ordered linked list of exoplanets.

Due Date:

You must submit an electronic copy of your source code through the dropbox in D2L by **Wednesday, December 9th by 11:00pm**.

Notes:

In this project, the only libraries you will use are `iostream`, `fstream`, `string`, and/or `cstring`.

Be sure to use good object-oriented design in this project. That includes appropriate use of encapsulation, inheritance, overloading, overriding, accessibility modifiers, etc.

Be sure to use good code documentation. This includes header comments for all classes and methods, explanatory comments for each section of code, meaningful variable and method names, consistent indentation, etc.

You may write your program from scratch or may start from programs for which the source code is freely available on the web or through other sources (such as friends or student organizations). If you do not start from scratch, you must give a complete and accurate accounting of where all of your code came from and indicate which parts are original or changed, and which you got from which other source. Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.