Project 3 – Hash Tables (Hash Maps)

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, merge, 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.	*5 points*
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.	*5 points*
4.	Store and exosystems in a linked list maintained in order, sorted by name. Remove entries from the exosystem linked list while maintaining its order. Integrate appropriate exception handling into classes that implement linked lists.	*5 points*
5.	Use a linked hash table with exoplanet names as keys for efficient retrieval of exoplanet data based on name.	*40 points*
6.	Provide a design document explaining and justifying implementation choices for your linked hash table.	*10 points*
7.	Use a sophisticated hash function and/or collision resolution method.	*10 bonus points*
►	Develop and use an appropriate design.	*15 points*
►	Use proper documentation and formatting.	*15 points*

Description:

For this project, you will revise and improve ExoPlanIt2.0 from Project 2 in one important way. You are encouraged to reuse and build on your code from Project 2. ExoPlanIt3.0 will have the same basic functionality as ExoPlanIt2.0 but it will have one major change "under the hood"– because it is believed that users will most often want to search for exoplanets by name, the list of exoplanet names will be used as keys to a hash table that stores exoplanet pointers. This will allow for exoplanets to be looked up by name in constant time, that is, in $\Theta(1)$ time. (Note that ExoPlanIt3.0 will still store the list of exosystems, and the lists of exoplanets within each system, using linked list data structures 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, ExoPlanIt3.0 will behave as described for ExoPlanIt2.0, except that there will be additional/refined/clarified data printing/display options, as follows.

'O' for ordered original output, will go through the linked list of exosystems, printing the summary data line on the exosystem itself (as with Project 0), followed by the lines for each exoplanet in the system (again, as with Project 0). ExoPlanIt3.0 will also leave two blank lines between the data for one system and the next (as with Project 0).

'W' for write, will write out the exoplanets in the current sorted order (based on the last field on which the data was sorted). Before each exoplanet is written out, the system summary data for its system will be written out. ExoPlanIt3.0 will also leave two blank lines between the data for one exoplanet and the data for the next, regardless of whether or not they are in the same system.

'L' for list, which will list (display) the exoplanets in order by name, regardless of the current sorting of the exoplanets based on other fields. As with option W, the system summary data for an exoplanet's associated system will be written out before the exoplanet's own data and ExoPlanIt3.0 will also leave two blank lines between the data for one exoplanet and the data for the next, regardless of whether or not they are in the same system.

Note that this means that the exoplanets themselves will be displayed in the same order with options O and L but the data will be clustered with O but not with L. That is, with O there will be one line of summary data for each system, followed by the data lines for all of the exoplanets in that system, followed by two blanks lines, followed by the system summary data line for the next system, etc. With L, there will be one system summary line, followed by one exoplanet data line (regardless of how many exoplanets are in that system), followed by two blank lines, followed by another system summary line (which, if the system has multiple exoplanets, may be the same as the previous system summary line), etc. In other words, the format for L is the same as the format for W but both O and L use an "alphabetical" (lexicographical) ordering, whereas W uses the other fields for ordering.

'D' for debug display that will list the exoplanets, one per line, in the order they are stored in the hash table, with each prepended with the bucket number (hash code) where it is stored. If a bucket is empty, the bucket number should still be displayed, followed by "NULL." If a bucket contains more than one item, each should be listed in its order within the bucket, one per line. If a bucket overflows somewhere besides another bucket, the overflow items should be prepended with "OVERFLOW:" and be displayed in their overflow order before proceeding to the next bucket. There should also be a blank line displayed between buckets. Note that this is thought of as a debug display as it is unlikely to be of use to an end user but may help you to debug your project. (The codes for the other options will remain unchanged, except that there will no longer be an option to sort by name, since that is now redundant with the hash table for searching by name and the list option for displaying the exoplanets by name.)

Implementation Issues:

In most areas, ExoPlanIt3.0 will be implemented just as was ExoPlanIt2.0. 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 list of exosystems and the lists of exoplanets within each system are stored as linked lists. The big implementation change will be the data structure used in the code to find exoplanets based on names. For ExoPlanIt3.0, you are no longer allowed to sort and search by name using the array, you need to use a hash table instead. For this reason, you should use a linked hash table where the linked list interwoven into the table is ordered by exoplanet name.

Beyond these requirements, you have a lot of freedom to choose hash table implementation details. For this reason, you should include an implementation document with your submission. Please make this a PDF file and name it "**implementation.pdf**" in your submission. In this document, you should describe the hash function you chose, the bucket size you chose for your hash table, the hash collision resolution strategy that you chose, and the load factor you chose, along with design justifications for each.

You are free to use any hash function you wish, so long as it results in hash codes within the bounds for your hash table. If you choose to implement a hash function that distributes keys close to uniformly and randomly throughout the table, you should note this in your documentation for the TA, so that he can assign you bonus points (up to 5 points, depending on the sophistication of the hash function used and your explanation of why its distribution is close to uniform and random).

You are free to use any bucket size for your hash table and any hash collision resolution strategy you wish. You should justify both of these design choices, noting that some bucket sizes may be more appropriate for some collision resolution strategies and vice versa. If you choose to implement a particularly sophisticated collision resolution strategy, you should note this in your documentation for the TA, so that he can assign you bonus points (up to 5 points, depending on the sophistication of the strategy and your explanation of why it is appropriate for your hash table).

Note that when you first create your hash table, you will know the amount of data it is to contain. This is because you can wait to create it until you have read in the first file of data. Similarly, when you merge or purge data, you can use your adjusted list to create a new hash table for the adjusted amount of data. However, you should not attempt to fill your hash table completely, because this will almost surely cause too many collisions and degrade its performance below that of a sorted array. Instead, you should choose an appropriate load factor for your table.

You should implement the display/printing options as follows.

O should be accomplished by traversing the ordered linked list of systems and within each system traversing that system's ordered link list of exoplanets.

W should be accomplished by traversing the sorted array of exoplanets.

L should be accomplished by traversing the linked list embedded within the hash table.

D should be accomplished by traversing the hash table in bucket order. If your buckets are larger than size one, each bucket should be traversed in order before proceeding to the next bucket. If your hash table overflows to anything besides other buckets, the overflow data structures should also be traversed in order before proceeding to the next bucket.

Due Date:

You must submit an electronic copy of your source code **and your implementation.pdf** file through the dropbox in D2L by **Wednesday, November 18th 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.