Project 2 – Linked Lists

Computer Science 2413 – Data Structures – Fall 2018
Due by 11:59 pm CST on Tuesday, 30 October 2018

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

User Request:

"Create a simple system to read, store, merge, purge, sort, search, and write NVRA data using a more complete array library, a linked list library, error checking, and user interaction."

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.	*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.	*5 points*
3	Integrate appropriate exception handling into the templated array class.	*5 points*
4	Efficiently sort the data based on the field specified by the user (using comparator).	*5 points*
5	Efficiently search the data based on the field specified by the user (using comparator).	*5 points*
6	While reading and merging NVRA data, use a linked list maintained in order by unique record IDs.	*15 points*
7	While purging, remove entries from the NVRA linked list while maintaining its order.	*10 points*
8	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.	*15 points*
9	Create and use an enumerator (iterator) for traversing the templated linked list.	*5 points*
10	Integrate appropriate exception handling into classes that implement linked lists.	*5 points*
11	Add user options to add additional functionality (merge and purge), prompt the user for additional file names, and provide error messages in case of file access errors.	*5 points*
12	Develop and use an appropriate design.	*10 points*
13	Use proper documentation and formatting.	*10 points*

Description:

For this project, you will revise and improve VoteR from Project 1 in several ways. You are encouraged to reuse and build on your code from Project 1. In addition to the functionality provided by VoteR 1.0, your new *VoteR 2.0* will allow users to merge and purge NVRA data from multiple files in its working memory throughout its use. VoteR 2.0 will also have a major change "under the hood"—rather than storing the list of NVRA records using an array that automatically resizes as entries are added, VoteR 2.0 will store the list of NVRA records using a linked list data structure when NVRA records are being added to and removed from VoteR 2.0's working memory. (Note that the list of NVRA records will still use a resizable array when the data is to be sorted and searched using binary search.)

Operational Issues:

VoteR 2.0 will behave as described for VoteR 1.0, except that rather than have separate data input and data manipulation loops, there will be a single user option loop that will allow the user to merge new data or purge existing data at any time by selecting 'm' for merge and 'p' for purge alongside the options present for output, sort, find, and quit. Also, there will be an additional data saving/display option 'r'. (Menu: "`Enter (o)utput, (s)ort, (f)ind, (m)erge, (p)urge, (r)ecords, or (q)uit: `").

The user will be prompted for a data file name when VoteR 2.0 is started ("`Enter data file name: `"), so that it has at least some data present before it enters the user option loop. As VoteR 2.0 reads that first data file, it will ensure that the list is maintained in order based on record ID number. If VoteR 2.0 has trouble accessing the data file named by the user, it should provide the user with an appropriate error message ("`File is not available.`"), then ask for a new file name. Similarly, if the file is available but no valid records are found in the file, VoteR 2.0 will give an error message indicating that no valid data was found ("`No valid records found.`") and again prompt for a file name. Only when at least one valid record is read in will VoteR 2.0 proceed to the user option loop. As with VoteR 1.0, if the user hits enter rather than entering a file name, VoteR 2.0 will exit.

If the user selects merge, VoteR 2.0 will prompt the user for the name of an NVRA data file using `cout` ("`Enter data file name: `"). VoteR 2.0 will then attempt to read that file, following the same rules regarding invalid data and duplicate record IDs within the same file as used in VoteR 1.0. All valid records with non-duplicate records IDs from this new file will be merged into the existing database, keeping the NVRA records sorted in order by record ID number. If any of these new records has a record ID equivalent to one already present in the file, the old record will be replaced by the new record.

If the user selects purge, VoteR 2.0 will likewise prompt the user for the name of a data file at the terminal prompt ("`Enter data file name: `"). VoteR 2.0 will then attempt to read that file, following the same rules regarding invalid data and duplicate record IDs within the same file as used in VoteR 1.0. However, in this case, it will purge (delete) from the database all entries with record IDs that match those of the records read from the file. If there are records in the purge file that do not appear in the existing database, these will simply cause no change to the data.

If VoteR 2.0 has trouble accessing the merge or purge file named by the user, it should an appropriate error message ("`File is not available.`") and repeat the prompt for the file name. If the user hits enter (and nothing else) at the prompt for a file name, VoteR 2.0 will return to the menu for the user option loop without having merged or purged any data, leaving all data structures unchanged.

At the end of each successful merge and purge operation for the NVRA linked list, VoteR 2.0 will replace the sorted array of NVRA data with a new array of NVRA data sorted by record ID by discarding the old NVRA array and copying the data from the NVRA linked list to a new NVRA data array. Note that reading data files for both merging and purging will have the potential to increase the number of data lines read and the number of valid records read.

If the user selects 'r' for record ID order, VoteR 2.0 will prompt for an output filename following the same process as for the output option ('o'), then will go through the linked list of NVRA data, printing each NVRA record in the order it is found in the linked list (which should be in order by record ID) using the same format and followed by the same summary line as for output ('o').

Implementation Issues:

In most areas, VoteR 2.0 will be implemented just as was VoteR 1.0. This includes how VoteR reads files (for the most part, see below) and outputs data, carries out user interaction via standard in and standard out, encapsulates C primitive arrays, and how exception handling is implemented for arrays and similar classes. The big implementation change will be the data structure used in the code to hold the list of NVRA records during initial reading, merging, and purging. For VoteR 2.0, you are no longer allowed to store the NVRA list in arrays while data is being added from a file or removed based on entries in a file—instead, VoteR 2.0 must use a linked list during those operations. The only time VoteR 2.0 will use arrays is for sorting and searching.

Note that VoteR 2.0 must keep the list of NVRA data sorted by record ID number as each entry is read in. This task must be accomplished by performing a linear search to find the appropriate place in the linked list to insert each entry and changing pointers as necessary to accommodate each entry. This is true for both the initial file and any merge files specified by the user.

Similarly, VoteR 2.0 purge files will be processed by linearly searching the NVRA linked list for each entry to delete and deleting it while retaining the order of the list.

Note that an alternative design to keeping these lists sorted during merging and purging would be to allow the lists to become unordered during these operations, then to sort them afterward. Give an analysis comparing the run times of these alternative designs and include this document in your project submission. Please make this a PDF file and name it "**VoteR2design.pdf**" in your submission.

Be sure to **use all provided code**, **use efficient mutator methods** (e.g., don't make new arrays unless doubling or halving the array), and **check whether memory is available** on the stack when using `new` in your `TemplatedArray` and `OULinkedList` classes and throw `ExceptionMemoryNotAvailable` if `new` returns `NULL`.

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.

Due Date:

You must submit an electronic copy of your VoteR 2.0 project to zyLabs and design document to the appropriate dropbox in Canvas by **11:59 pm CST on Tuesday, 30 October 2018**.