# Project 1 – Searching and Sorting

*Computer Science 2413 – Data Structures – Fall 2019*

*Due by 11:59 pm CST on Monday, 7 October 2019*

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

## *User Request:*

*"Create a simple system to read, store, write, sort, and search drilling data using a more complete array library, error checking, and user interaction."*

## *Objectives:*

| | | |
|---|---|---|
| 1 | Use C++ file I/O to read and write files while using C++ standard I/O (`cin`, `cout`) for user interaction, using appropriate exception handling. | *15 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 in resizable array and drilling record classes. | *5 points* |
| 4 | Efficiently sort the data using comparator based on the field specified by the user. | *35 points* |
| 5 | Efficiently search the data using comparator based on the field specified by the user. | *20 points* |
| 6 | Develop and use an appropriate design. | *10 points* |
| 7 | Use proper documentation and formatting. | *10 points* |

## *Description:*

For this project, you will revise and improve Driller from your labs in several ways. You are encouraged to reuse and build on your code from your labs. In addition to the functionality provided by Driller 0.4, your new *Driller 1.0* will take user input that allows users to specify the files in which data is stored and to sort and search for entries based on various data fields.

## *Operational Issues:*

Driller1.0 will read drilling data files via C++ file I/O. The names of the data files will be specified by the user using standard input. When Driller 1.0 starts, it will enter a data input loop, prompting the user for the name of a data file ("`Enter data file name: `") and waiting for the user to type a file name and hit enter. If the user enters the name of an available data file, Driller 1.0 will open the file using C++ file I/O and read the data. If the user enters the name of a file that is not accessible, Driller 1.0 will report the error to the user ("`File is not available.`") and continue in the loop, repeating the prompt and waiting again for a file name. If the user hits enter without entering anything else, Driller 1.0 will exit the data input loop. If no data has been read in when Driller 1.0 exits the data input loop, Driller 1.0 will exit. Otherwise, Driller 1.0 will move on to a data manipulation loop (see below).

The data files will be organized as they were in the labs. However, for this project, the drilling data may be spread across multiple files.

When Driller reads in each file, it must make sure the data in that file is internally consistent and alert the user to any errors encountered. This will be almost identical to what was described in Lab 1—all

records must have the same date, all time stamps must be unique, and all numeric data values must be greater than zero. However, in Driller 1.0, if a data line has invalid numeric data (values of zero or less), it's time stamp will *not* be added to the list of unique time stamps read in. Instead, only completely valid data lines will have their time stamps count as having been read in. This means that a separate list of time stamps does not need to be maintained. Instead, Driller 1.0 may use the valid data stored in its resizable array to check for duplicate time stamps.

Moreover, when Driller reads any subsequent file, it will check to ensure that the date of the first data line matches the date of original file. If the dates do not match, Driller will close the new file and inform the user of the problem ("`Date mismatch; file closed.`") before asking the user for a new file ("`Enter data file name: `") and continuing with the data input loop. If a drilling record with the same time stamp appears in more than one data file, Driller will retain the data read **most recently** (note the difference from the labs, although Driller 1.0 will still reject duplicates time stamps within the same file). As each file is read in, Driller must update internal tallies of the total number of drilling data lines it attempted to read, the total number of valid drilling records read in, and the total number of drilling records currently stored in memory. (Note that these totals may be the same, but they may be different, as some data lines may be invalid and some valid drilling data lines may have the same date and time.)

After reading and storing all drilling data from all specified files, Driller will enter a data manipulation loop (as mentioned above). In this loop, Driller 1.0 will prompt the user with four options: 'o' for output, 's' for sort, 'f' for find, and 'q' for quit ("`Enter (o)utput, (s)ort, (f)ind, or (q)uit: `").

If the user selects output, Driller 1.0 will prompt the user for a filename ("`Enter output file name: `") and read it in from the console. If the user hits enter without specifying a filename, Driller 1.0 will send output to standard out. Otherwise, it will attempt to open the specified file for writing. If the user enters the name of a file that is not accessible, Driller 1.0 will report the error ("`File is not available.`") and repeat the prompt for the file name. If the file is accessible or the user has specified the standard out, Driller 1.0 will print out the drilling data in whichever order it is presently ordered, followed by a row showing the internal tallies, for example:

```
Data lines read: 15847; Valid drilling records: 13347; Drilling records in memory: 11879
```

Other than this trailing row of tallies, and possible differences in the order of the rows, the data output format of Driller 1.0 will exactly match the output format used by Driller 0.1 through Driller 0.4.

If the user selects sort, Driller 1.0 will prompt the user for the data field on which to sort the records ("`Enter sort field (0-17): `"). The user may select any numeric value from 0 to 17 (inclusive), which corresponds to the column number on which to sort. If the user enters an invalid value for the data field on which to sort, Driller 1.0 will return to the data manipulation loop. If the user selects to sort by any valid data field, Driller 1.0 will sort the data based on that field.

If the user selects find from the data manipulation loop, Driller 1.0 will prompt the user for the data field on which to search ("`Enter search field (0-17): `"). As with sort, the user may select any numeric value from 0 to 17 (inclusive) and if the user selects any other value, Driller 1.0 will return to the data manipulation loop.

If the user chooses to find by any numeric column, Driller 1.0 will prompt the user for the numeric value on which to search ("`Enter positive field value: `"). If the user enters any text (doesn't just hit enter), Driller 1.0 will ensure that the value entered is valid (can be converted to a positive number) and, if it is, search for the given value. If the user entry is invalid, Driller will return to the data manipulation loop.

If the user chooses to find by any other valid field, Driller 1.0 will prompt the user for the value on which to search ("`Enter exact text on which to search: `"). If the user enters any text (doesn't just hit

enter), Driller 1.0 will search for the given value but does not need to perform any validity check on the input data. If the user just hits enter without giving a search string, Driller 1.0 will return to the data manipulation loop.

If Driller 1.0 finds one or more fields exactly matching a user request, Driller 1.0 should display for the user (by sending to standard out) all the corresponding rows, in the same format as when printing the data on all drillings, but with a trailing row that indicates the number of drilling records found in the format shown in the following example:

```
Drilling records found: 1897.
```

If no matching records are found, Driller 1.0 should simply send to standard out the line indicating that the number of drilling records found was zero:

```
Drilling records found: 0.
```

If the user selects quit from the data manipulation loop, Driller 1.0 should display a friendly parting message ("`Thanks for using Driller.`") and exit.

### *Implementation Issues:*

Given that the data may be read in from multiple files and that drilling timestamps may be repeated across files but only the last read data for a given date and time should be retained, Driller 1.0 will need to search for each drilling date and time before the drilling data is inserted into the data array. Consider how to most efficiently maintain the array and search for each date and time as its data is read in. Here you should assume that the number of duplicates is much less than (≪) than the number of data items. Also, you should assume that if the user has not asked to sort the data by any column, then it should be sorted by time when it first leaves the data input loop and moves to the data manipulation loop. **You will create simple design document that explains and justifies your design choices with respect to these issues.** Please make this a PDF file and name it "**Driller1design.pdf**" in your submission.

If the data is sorted on a different field from the one on which the user is searching, Driller 1.0 will not be able to conduct a binary search for the data. In this case, Driller 1.0 should conduct a linear search instead. On the contrary, if the data is sorted on the field on which the user is searching, Driller 1.0 should conduct a binary search. Note that you will have to modify the binary search in an important way if the data in a given field is not guaranteed to be unique. **Your design document will explain and justify how you modified binary search to account for potential duplicates.** Further note that the only data field for which data is guaranteed to be unique is the combined date and time. Here you should assume that the number of repeated data values in any column is much less than (≪) than the number of data items.

Note that if the binary search does not find the item searched for, it should return a negative value indicating at which index the item should be inserted. This is a standard approach to binary search algorithms, because it means that the same algorithm can be used for finding items in an array when they are present and also for determining where to insert items in the array if they are not already present. The returned negative value should indicate the appropriate insertion index as –(index+1). For example, if the appropriate insertion index is 0 (moving the item already at index 0 to index 1), then the binary search should return the value –1. (Note that the returned value must be offset by 1 from the desired insertion index because there is no way to return –0.)

Because you now have a templated, resizable array class and methods for efficiently sorting it and searching it, you may no longer use similar data structures from other sources, including the C++ Standard Template Library (including `vector`, `set`, and others), except for parsing input lines.

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 `ResizableArray` class and throw `ExceptionMemoryNotAvailable` if `new` returns `NULL`. Also, be sure to throw `ExceptionIndexOutOfRange` in member functions of `ResizableArray` and `DrillingRecord` that deal with indices. (See their header files for details.)

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(s). 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 Driller 1.0 project to zyLabs, submit the score from zyLabs to Canvas, **and submit your design document to the appropriate dropbox in Canvas** by **11:59 pm CST on Monday, 7 October 2019**.