**This assignment is used to distinguish you from others. It will test your true potential. If you score an average score on this assignment you will be invited to the interview. The starter code is provided** [**here**](https://drive.google.com/file/d/0B1ajcFr2rfq0UzFxX0doZGFOcGM/view?usp=sharing)**. The interview will start from tomorrow. The time and method for interview will be emailed to selected candidates.**

**After finishing the assignment send your code in zip file with name FirstName\_LastName\_SkypeId\_C\_Developer.zip to the email** [**assignment@kandara.tech**](mailto:assignment@kandara.tech)

Write a small data-base (d-base) management program in C++ to process records containing information of several different types.

* Maintaining a bibliography for a research paper or a list of titles for a bookstore or library. Components of records might then include strings for the author's surname, first name, book or journal title; the copyright date (int); pages used (firstPage, lastPage: int); perhaps the purchase price (float); perhaps the binding (bool paperback). The components mentioned are examples -- perhaps not all will be appropriate, and perhaps you will want to use others.

What you are required to write is a program showing that you can use C++ classes, including "methods" - functions that are part of a class. Your program should use at least one type of record that has several components of different types. You should demonstrate that you can access values stored in record-structured variables by both the "dot" or "period" notation, and by using the complete record, for example by assigning a record-structured variable's value to another such variable (this might come up in a sort, for example), or, where appropriate, passing a complete record as a parameter to a subprogram.

**SUGGESTED DATA STRUCTURES**

* One good way to handle this project is with a "linked list" of records, using "pointers" and C++'s "new" and "delete" operations. An important advantage of this method is that it enables you to write your program for a data base of arbitrary size (limited only by available memory).
* However, it is possible that you won't feel you have mastered such a data structure in time for you to write your program in this fashion. As an alternative, you may use an array of recType, where "recType" (substitute an identifier that suits your program) is a class you define for the records in your list of data. HINT: In this case, you should find it useful to include a field

*bool inUse;*

that should be *true* while the record holds current (good) data, *false* otherwise. For example, setting this field false is an easy way to delete a record, provided you check this field before all other operations.

Hints:

* + To insert a new record into such an array, search for an entry whose *inUse* field is *false*. If such a record is found, use it for the data of the new record and set the *inUse* field *true*.
  + To delete a record in such an array, perform a find/search operation to locate the record to be deleted. If the record is found, set its *inUse* field *false*.
* You would be wise to develop a class for the basic record, and another class for a list (e.g., array) of such records. An important question for you to deal with will then be: for a given operation, is it a single-record operation that should be a method of the basic record class, or is it a list operation that should be a method of the list class, or some combination of both?
* Use separate source code files for each class or each set of closely related classes ("header" files with extension ".h" -- without the quotation marks) (if you create closely related classes, you may use the same file for their code), and another source code file for your driver program (".cc" or ".cpp" extension). These are stitched together with appropriate "#include" directives.

Your program should be "menu-driven," with the menu including (but not limited to) options to add a record, change a record (in which case perhaps another menu offering a choice of fields should be presented), delete a record, print the information stored in a record, compute a summary statistic (such as a total, average, maximum, or minimum), or print the entire list of records -- SORTED by an appropriate field. Thus, you should be able to sort your records by at least one field. You're welcome to discuss sorting with me.

You should read initial data records from an external data file. This saves you a great deal of time in running and testing your program, as you won't have to type in data records to give your program an initial database each time you run the program. You should save the final collection of records in the SAME file at the end of your run (as an updated database). Note you should first close the file as an input file before you open it as an output file.

The output file should be formatted to look much as it did at the start of the program, when it was used for input - format your output file for your ease in using it as an input file. It's your screen output that should be formatted for the human reader.

Following things are suggested to pass the skill test

* As mentioned above, use pointers, the "new" and "delete" functions, and the special constant "NULL" to handle your list of records as a linked list rather than as an array. However, YOU ARE HEREBY WARNED that pointers are tricky to learn to use, and you may not have time to master them adequately for this project.
* Use unions to provide record variants, so that the same data structure (*e.g.*, the same array or linked list of records) can contain different kinds of records. For example, you might want a pitcher's record to have a different structure than a non-pitcher's if you are keeping baseball statistics, yet treat both as examples of a baseball player record.
* Use of recursive functions, *e.g.*, to sort or to search a sorted list. If you use recursion, it will be expected that yours is a "good" use, and not simply a form of looping better handled by *do*, *for*, or *while* loops.

As usual, style counts. There should be no global passage of variables between program units -- information to be shared should be passed through parameter lists. Similarly, consts and types not used globally should be declared within the subprogram(s) using them. "Magic values" should be declared const. Functions should be explained in comments.