LARP: Linked Advising Records Program

This assignment will modify and extend DARP. All of the specified functionality of DARP will be retained. There are two major extensions.

First, the student data structure will be extended. LARP will use a linked list of dynamically allocated nodes to store multiple student advising records. The list may be either singly or doubly linked, whichever you decide is most appropriate for your implementation. In order to receive full credit, the list must be fully encapsulated using a C++ class, and the list nodes must themselves be implemented using a node class. In addition, the student record stored in each list node must be encapsulated; but it is acceptable for the student records to be implemented as struct variables (as the single student record was in SARP and DARP). The course records for each student will be stored as a dynamic array within the corresponding student record.

Second, in order to make the program more useful, you will add the ability to read, delete and order student records from/to the students linked list. This will require changes to the input and output specifications and the addition of three new actions:

```
read  <SSN> <StudentDataFile>

del   <SSN>
```

and

```
ord   <FieldSpecifier>
```

LARP will process command-line arguments in almost the same way as DARP.

```
DARP <DatabaseFileName> <DatabaseActionsFileName>
```

The exception being that the first DARP command-line argument `<InitialStudentFileName>` will be replaced by `<DatabaseFileName>`. This is an optional command-line argument which will represent a previously saved database file. The database file format is still unspecified, but must now be extended to handle multiple student records and course information. When an optional `<DatabaseFileName>` is specified, by default the student records must be inserted in order by their SSNs.

Input file descriptions:

The actions file will have almost the same syntax as for DARP. Note that use of hard-coded file names will annoy the person evaluating your program, and you will be charged points for that annoyance. Sample input files will be posted on the website shortly.

Each line of the actions file will contain one of the commands described in the SARP and DARP specifications, or one of the commands described below. As before, commands are case-sensitive and take a fixed number of arguments. The command names will be valid and each command will include the correct number of arguments. Command arguments will be tab-delimited.

```
read  <SSN> <StudentDataFile>
```

> A `read` command causes the specified student data file to be input and stored in the linked list database. The `<StudentDataFile>` will have the same format as in the previous programs, storing a single student's academic and course information.

```
del   <SSN>
```

A `del` command causes all of the specified student's information to be removed from the linked list database.

`ord <FieldSpecifier>`

This causes the database list of student records to be sorted into ascending order by the specified field. The FieldSpecifier must be one of: `SSN`, or `Name`. If a `ord` instruction specifies an invalid FieldSpecifier, the list will not be modified. You should use the selection sort algorithm. For the `Name` FieldSpecifier the sort should be performed using the concatenation of both the student's last and first names.

Two of the previous action file commands will be modified slightly to account for the handling of multiple students. The `drop` and `sort` commands will be changed to accept a `<SSN>` as their first argument.

`drop <SSN> <IndexNumber>`

This causes the deletion of the course record of student `<SSN>` with the indicated <IndexNumber> number. If a `drop` instruction specifies the number of an item that's not in the course list, the list will not be modified.

`sort <SSN> <FieldSpecifier>`

This causes the course list of student `<SSN>` to be sorted into ascending order by the specified field. The FieldSpecifier must be one of: `Index`, or `Course`. If a `sort` instruction specifies an invalid FieldSpecifier, the list will not be modified. You should use the selection sort algorithm. For the `Course` FieldSpecifier the sort should be performed on both the `Department` and `Course` number fields.

Output description and sample:

Output data resulting from a `dump` command must be written to a file named `dbase.list` — use of any other output file name **will** annoy the person evaluating your program and you will be charged points for that annoyance. The format of `dump` output should be the same as for SARP, except that you must now output multiple student records and course lists. Sample output files will also be posted on the course website shortly.

Linked List:

You are **required** to use a linked list to store the student's information. Your implementation of this list will be examined. In order to receive full credit, you must implement the nodes using a well-designed node class and the linked list itself must be encapsulated using a well-designed list class. In addition, the student records must be encapsulated so that the data they store is logically separated from the node pointers.

Note this mimics the behavior of a C++ STL (Standard Template Library) list object. You are specifically forbidden to use any C++ STL list objects in this program, or any other sort of predefined dynamic list type. You may base your implementation on the linked list implementations shown in the textbook, or those shown in the notes and lectures. You may not use linked list code from any other source. Advanced C++ OOP constructs and concepts, (e.g., inheritance, virtual functions, templates, etc.), not covered in CS 1704 are also not to be used in this program. Violating these restrictions would remove major points of this assignment and will certainly result in a large deduction.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the requirements for documentation and coding given in the DARP specification are still in effect. In addition:

Documentation:

- You must describe the purpose of each of your classes in a header comment that precedes the class declaration.
- You must document each data member and function member of your classes, both in the header file containing the class declaration and in the corresponding source file containing the implementation. The header file does not have to contain full documentation for each function, but the purpose of each function should be described there.

Coding:

- You must separate the interface of each of your classes from its implementation by placing each class declaration (interface) in its own header file and the implementation of that class in a corresponding source file. The name of the class should be used as the name of the header and source files.
- You must protect access to your data by making **all** data members of your classes private.
- When a node is removed from your linked list, you must dispose of it properly by using `delete`.
- When you discard your student database list to load an existing one from a file, you must `delete` every node and dynamic array in the old list so that your program does not waste memory.
- Memory leaks that might affect the execution or functionality of your program will be penalized. In fact, you should avoid memory leaks altogether.

Deliverables:

You must submit an interim design document, to the Curator, no later than midnight Tuesday March 7. The required format of this design document will be specified shortly. This design document must indicate your current design plans for LARP. It is expected that your final design will differ from the interim design. Nevertheless, your interim design should be relatively complete.

Your final project submission must include the following (and absolutely no other files):

- all source code (`*.cpp` and `*.h` files) comprising your project
- a revised design document reflecting the final design of your project, at the time of submission; this must either be in a format that can be viewed in MS Word or be a PDF file.
- one set of input files, named `<student>.data` and `actions.data`, and the corresponding final dump `dbase.list`, and the corresponding saved database file, named `LARP.db`
- a brief ASCII text readme file, named `readme.txt`, with any special execution instructions
- either the MS Visual C++ `.dsp` and `.dsw` files, or a UNIX makefile, as appropriate

Submissions will be archived, but not scored, by the Curator. You will submit your project as an archive file in one of two formats. For Windows users, submit a zipped archive containing the items listed above. (The program WinZip is very easy to use and is available from the University Computing Services website: `http://www.ucs.vt.edu/`) For UNIX users, submit a gzipped tar file containing the items listed above.

Note that omitting files from your archive is a classic error. Once you've created your project archive, copy the file to a new location, unzip it, attempt a build and then test the resulting executable. Submitting an incomplete copy of your project will delay evaluation and result in a loss of points.

Evaluation:

LARP will be subjected to runtime testing by the TAs, who will also score your implementation for adherence to the specified programming standards. Your score will depend on the quality of your design documentation, the amount of change from your interim to your final design, the quality of your coding and internal documentation, and the correctness

and completeness of the output your program produces during runtime testing. The relative weights of these factors may be announced later.

Testing:

At minimum, you should test your program on **all** the posted input/output examples given along with this specification. You could make up and try additional input files as well; of course, you'll have to determine by hand what the correct output would be.

Submitting your project:

You will be allowed to make up to five submissions of LARP to the Curator. You may use the JSP Server client, or you may use either the application or applet version of the Java client. In any case, begin at:

http://spasm.cs.vt.edu:8080/curator/

We recommend using the Java application version of the client. If you use either of the Java clients, instructions for submitting your project archive are available in the *Student Guide* at the Curator Project Homepage:

http://ei.cs.vt.edu/~eags/Curator.html

Read the instructions carefully.

Note well: your last submission will be graded.

Pledge:

Each of your project submissions to the Curator System must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment of the `cpp` file containing `main()`:

```
//    On my honor:
//
//    - I have not discussed the C++ language code in my program with
//      anyone other than my instructor or the teaching assistants
//      assigned to this course.
//
//    - I have not used C++ language code obtained from another student,
//      or any other unauthorized source, either modified or unmodified.
//
//    - If any C++ language code or documentation used in my program
//      was obtained from another source, such as a text book or course
//      notes, that has been clearly noted with a proper citation in
//      the comments of my program.
//
//    - I have not designed this program in such a way as to defeat or
//      interfere with the normal operation of the Curator Server.
```

Failure to include this pledge in a submission is a violation of the Honor Code.