## LAMS: Linked Appraisal Movie System

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

First, the movie data structure will be changed. LAMS will use a linked list of dynamically allocated nodes to store multiple movie records. The list may be either a singly or doubly linked list. Choose whichever you decide is most appropriate for your implementation, (a double linked-list is recommended). 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 movie record stored in each list node must be encapsulated in a class, (as required by the DAMS specification).

Second, in order to make the program more useful, you will add the ability to modify (replace) movie records from/to the movie linked list and manage the availability of movies. This will require changes to the input and output specifications and the addition of three new actions:

```
modify      <Title> <Director> <Release Year> <Performers> <Rating> <Stars>
            <Stock> <Rental> <Serial> <Review>
```

This causes the replacement of the corresponding movie record in the database list. Replacement should place the modified record in the same location with respect to the current sort ordering of the list. The movie list will be searched for a movie record that matches the modify <`Serial`> number. If a matching <`Serial`> number is not found in the list, the list will not be modified. Note that due to page limitations, (not file line size), the last four arguments of the add command description above have wrapped onto the next line. In the actual `actions.data` input file they will all be tab separated on the same line.

```
stock <Serial>    <int>
```

This causes the stock count of the corresponding movie record in the database list to be increased or decreased by the specified integer value argument. (Positive values indicate that more copies of the movie have been purchased. Negative values indicate that movie copies have been removed from circulation for various reasons). If a corresponding (matching) <Serial> number is not found in the list, the command is ignored. If a stock command causes the stock count to drop to zero and one or more copies are currently rented this would mean that when the movie(s) are returned they are either trashed or placed on sale but are no longer available for rental. If a Stock command would result in a negative stock value that would correspond to allowing a store to remove from circulation (sale/trash) a number of copies of a movie that doesn't exist. (Not physically possible.) Thus for this case consider the command an error and ignore it.

```
rental <Serial>   <+1 | -1>
```

This action indicates that a copy of the movie has been rented (-1) or returned (+1). It causes an increment or decrement in the rental field of the corresponding movie record in the database. Note that movies cannot be returned if all copies of a movie are available for rental. Likewise, a movie cannot be checked out if all copies of the movie have already been rented. In these cases, (and the following case), no action is taken. If a corresponding (matching) <`Serial`> number is not located in the list, the rental command is not processed.

LAMS will process command-line arguments in almost the same way as DAMS. Two possible invocations of LAMS will be possible.

```
LAMS <InitialMovieDataFileName>
```

or

```
LAMS <DatabaseFileName> <DatabaseActionsFileName>
```

In the first invocation, (with only one LAMS argument), LAMS will read the `<InitialMovieDataFileName>` and convert it into the LAMS proprietary database file format. A default name for the database file "`dbase.db`" must be used.

When an `<InitialMovieDataFileName>` is specified, by default, the movie records must be inserted in order by their Serial numbers. The second invocation, (with two LAMS arguments), specifies that the first LAMS command-line argument `<DatabaseFileName>`, represents a previously saved database file. The database file format is still unspecified.

## Input file descriptions:

The actions file will have the same syntax as for SAMS and DAMS. 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 SAMS and DAMS specifications, or one of the commands described above. 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.

## 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** result in you being charged points for not adhering to specifications. The format of `dump` output should be the same as for SAMS. 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 movie 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 movie 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 SAMS and DAMS specifications 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 movie database list to load an existing one from a file, you must `delete` every node 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 severely penalized.

## Deliverables:

You must submit an interim design document, to the Curator, no later than midnight **Friday Oct. 13**. This design document must indicate your current design plans for LAMS. 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 posted input text files, the corresponding final dump `dbase.list` text file and the corresponding saved database text file, named `LAMS.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. **Also note** that including unnecessary files is also a classic error. Visual C++ users: do not zip up the **debug** or **release** subdirectories!

## Evaluation:

`LAMS` will be subjected to runtime testing by the TAs, who will 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.

## Testing:

At minimum, you should test your program on **all** the posted input/output examples. 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 submit your project archive to the Curator System, as described here. `LAMS` will be subjected to runtime testing by the TAs, who will also score your implementation for adherence to the specified programming standards. You will be allowed to make up to five submissions of `LAMS` to the Curator. 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.**