LARS: Linked Automobile Record System

This assignment will modify and extend DARS. All of the previous functionality of SARS, CARS and DARS will be retained, unless explicitly removed or modified in this specification. There are two major extensions.

First, the DARS automobile data structure will be changed. LARS will use a **double linked-list** of dynamically allocated nodes to store multiple automobile objects. 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.

Second, in order to make the program slightly more useful, you will add a histogram command. This will create a horizontal histogram of the dealer's average automobile price currently stored in the list (see input/output file descriptions below). In addition, the program will provide the capability of dumping the list ordered on different fields. This will require slight changes to the input/output specifications, the addition of one new action and the modification of one existing action:

```
histogram
```

and

```
dump  <FieldSpecifier>
```

LARS will process command-line arguments in almost the same way as DARS.

```
LARS <InitialAutomobileSystemDataFileName>
```

or

```
LARS <AutomobileSystemDatabaseFileName> <DatabaseActionsFileName>
```

The change being that if only one command-line argument `<InitialAutomobileSystemDataFileName>` is present, (representing an initial automobile data file), it will be input and automatically saved as correspondingly named `<InitialAutomobileSystemDataFileName>.cdb` database file. (No actions file will be processed in this case.) The database file format is still unspecified. If two command-line arguments are present the first will represent a previously saved database file.

Input file descriptions:

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

Each line of the actions file will contain one of the commands described in the SARS, CARS and DARS specifications, or one of the commands described below. As before, commands are case-sensitive and take a fixed number of arguments. It may be assumed that the command names will be valid and each command will include the correct number of arguments. Command arguments will be tab-delimited.

```
histogram
```

> A `histogram` command causes the current automobile objects, stored in the linked-list database, to be traversed, the average price of each different dealers' automobiles to be determined and a proportional horizontal histogram of the average automobile prices to be output to the dump file. The bar representing the automobile prices will begin in column 11. It will be preceded by the dealer's name in columns 1-9, (truncated if necessary), and a colon in column 10. The maximum length for a bar will be 70 characters. The bar for the dealer with the greatest average automobile price will be exactly 70 characters, with the other price bars output proportionally. The characters representing the bar for a dealer will be the characters of the dealer 'sname itself, (see the output

section below). For ease of histogram production, it may be assumed that no more than 10 different dealers will exist in the database at any given time.

`dump <FieldSpecifier>`

This causes the database list of automobile records to be printed to the `dbase.txt` output file in ascending order by the specified field. As previously, each automobile record output will be numbered starting at zero. However, for LARS no maximum storage size for the database list will be output. The FieldSpecifier must be one of: VIN, `Dealer, Price`. These represent ordering upon the corresponding fields. If a `dump` instruction includes an invalid FieldSpecifier, the list will not be output. You are free to use whatever sort algorithm you wish.

Output description and sample:

As before, output data resulting from a `dump` command must be written to a file named `dbase.txt` — use of any other output file name **will** result in a large deduction. The format of `dump` output should be the same as previous programs, except that the `[XXX] END` output at the end of database dump will no longer be included. Sample output files will also be posted on the course website shortly.

Given the sample database file contents output dump below, the corresponding histogram command output would be:

```
Programmer:  Dwight Barnette
Linked Automobile Records System
Update:           1BD2D8C89Y1285739   *MISSING*
Update:     004   1MMD4C629H1142630    3800
Find:             1BD2D8C89Y1285739   *MISSING*
Find:       004   Shelob MM            3800
Dealer:           Hud's Son           *MISSING*
Dealer:     003   Shelob MM           11351.67
_____________________________________________________________________________
IDX VIN               Dealer       Manufacturer Model           Year    Price
000 14B2D6C1921102113 Pooka GM     Buick        LeSabre         2002     22250.00
001 1BD2D8C89Y1285738 Pooka GM     Dodge        Intrepid        2000     19990.00
002 1CC4D4CC9Y0325751 Shelob MM    Chrysler     PT Cruiser      2000     24500.00
003 1FH2D8C19I2129562 Shelob MM    Lincoln      Continental     1987      5750.00
004 1MMD4C629H1142630 Shelob MM    Mercury      Mystique        1986      3400.00
005 1S4WD4449V4268422 Pooka GM     Subaru       Outback         1997     16750.00
[010] END
_____________________________________________________________________________
Years:      1988...1992               *Zero*
Years:      1998...2002         3     22246.67
_____________________________________________________________________________
                      Automobile Price Histogram

Pooka GM :Pooka GMPooka GMPooka GMPooka GMPooka GMPooka GMPooka GMPooka GMPooka
Shelob MM:Shelob MMShelob MMShelob MMShelob MMShel
_____________________________________________________________________________
```

Linked List:

You are **required** to use a double linked-list to store the automobile system 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 automobile system data 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/static list type. You may base your implementation on the linked list implementations shown in the textbook, or those shown in the notes and lectures. (The lecture notes implementation is recommended, as the Deitel implementation node class is an instantiation of a template class. If you elect to base your implementation on the Deitel code you must modify it **not** to use templates.) 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 previous 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.

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 automobile system 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 penalized. In fact, you should avoid memory leaks altogether.

Interim Design:

You will produce an interim design for LARS, and represent that design in a modular structure chart. The structure chart must indicate your design plans for LARS. It is expected that your final design will differ from the interim design. Nevertheless, your interim design should be relatively complete. If the interim design is incomplete or if the differences between your interim design and your final code are excessive, you will be penalized. That means that you should take the production of the interim design seriously, but **not** that you should avoid changes that would improve your final implementation of LARS. In addition to the interim design, you must also submit a stepwise refinement implementation plan. This must be no longer than one page and must follow the structure chart in the submitted MS Word.doc file. The implementation plan must detail your planned order of implementation of the LARS systems functions. The dates by which the functions will be completed, tested and integrated, (along with a brief description of the resulting system capability), must be present in the plan. You must submit this interim design/plan, to the Curator System, no later than midnight **Monday, March 25**. Submit a MS Word.doc file. Do **NOT** compress, (zip), the interim design/plan submission! The file must contain no bit-map drawings; only vector-structured graphic drawings and text will be accepted. The file size must be no more than 300K.

Testing:

Obviously, you should be certain that your program produces correct output on all given sample data. However, verifying that your program produces correct results on a few test cases does not constitute a satisfactory testing regimen. You could make up and try additional input files as well; of course, you'll have to determine what the correct output would be.

Deliverables:

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

- all source code (`*.cpp` and `*.h` files) comprising your project
- The MS Visual C++ project files (.dsp and .dsw). Do **NOT** submit the debug/release project subdirectories or an executable.exe file.

- a **revised** modular structure chart reflecting the final design of your project, at the time of submission; this must in the same MS Word format as the interim chart.
- one set of input files, named `AreaData.txt` and `Actions.txt,` and the corresponding final dump `dbase.txt`, and the corresponding saved database file(s)`.cdb.`
- a brief ASCII text readme file, named `readme.txt`, with any special execution instructions

Submissions will be archived, but not scored, by the Curator System. You will submit your project as an archive file in a zipped archive containing the items listed above. (The shareware program WinZip is very easy to use and is available from the Computing Services website: http://www.ucs.vt.edu/)

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 may delay its evaluation and **will** result in a substantial loss of points. In particular, if you omit a source file necessary to compile your program, you **will** be allowed to supply that file; however, we will then apply a **late penalty** corresponding to the date that you have provided a complete copy for evaluation. There will be **no exceptions** to this penalty. **Also note** that including unnecessary files is also a classic error. Visual C++ users: do **not** zip up the **debug** subdirectory!

Submitting your project archive:

You will submit your project archive to the Curator System, as described above. All submissions for LARS must be made by midnight Monday April 15th. There will be **NO late submissions** accepted for LARS. LARS will be subjected to runtime testing by the TAs, who will also score your implementation for adherence to the specified programming standards. Demonstration time slot signup sheets will be made available in the CS lab. An announcement will be posted when they are available. Students will only be allowed to schedule and perform one demonstration. TA/student demonstration assignments will be posted on the course Web site. You will be allowed to make up to five submissions of LARS to the Curator. Note well: **your last submission will be graded.** There are no exceptions to this policy! Changes made to code during a demonstration will be heavily penalized.

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.