LCIS: Linked Census Information System

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

First, the CIS area data structure will be changed. LCIS will use a **double linked-list** of dynamically allocated nodes to store multiple CIS area 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 area populations currently stored in the list (see input/output file descriptions below). In addition, the program will provide the capability of ordering the list on different fields. This will require changes to the input and output specifications and the addition of one new action and the inclusion of one old action:

`histogram`

and

`sort  <FieldSpecifier>`

LCIS will process command-line arguments in almost the same way as DCIS.

`LCIS <InitialCISAreaDataFileName>`

or

`LCIS <DatabaseFileName> <DatabaseActionsFileName>`

The change being that if only one command-line argument `<InitialCISAreaDataFileName>` is present it will represent a `initial CIS area data file` that will be input and automatically saved as correspondingly named `<InitialCISAreaDataFileName>.cis` 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 DCIS. 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 soon.

Each line of the actions file will contain one of the commands described in the SCIS, CCIS and DCIS 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 area objects, stored in the linked-list database, to be traversed and a proportional horizontal histogram to be output to the dump file. The bar representing the area population will begin in column 10. (It will be preceded by the FIPS number of the area in columns 1-5 and a colon in column 6.) The maximum length for a bar will be 70 characters. The bar for the area with the largest population will be exactly 70 characters, with the other area bars output proportionally. The characters representing the bar for an area will be the characters of the area's name itself, all capitalized, (see the output section below).

`sort <FieldSpecifier>`

This causes the database list of CIS area records to be sorted into ascending order by the specified field. The FieldSpecifier must be one of: `FIPS, Pop`. These represent ordering upon the `FIPS`, or population fields. If an `sort` instruction includes an invalid FieldSpecifier, the list will not be modified. You must use the selection sort algorithm.

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** 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 previous programs, except that the `xx> MAX CIS STORAGE` 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 Census Information System
```

Area	State	FIPS	SqrMiles	POP90	FEMALE90	MALE90	WHITE90	BLACK90	HISP90	OTHER90
1. Alexandria	VA	51510	15.847	111183	58442	52741	76789	24339	10778	4785
2. Bedford	VA	51515	8.574	6073	3220	2853	4691	1338	53	0
3. Bristol	VA	51520	11.590	18426	10202	8224	17240	1063	64	8
4. Charlottesville	VA	51540	11.759	40341	21406	18935	30684	8561	476	94
5. Chesapeake	VA	51550	351.980	151976	77509	74467	107399	41662	1913	594
6. Colonial Height	VA	51570	9.328	16064	8554	7510	15502	129	161	70
7. Covington	VA	51580	5.532	6991	3729	3262	5953	969	27	44
8. Danville	VA	51590	17.454	53056	28864	24192	33247	19431	276	25

```
                                CIS Histogram:

51510:   ALEXANDRIAALEXANDRIAALEXANDRIAALEXANDRIAALEXANDRIAA
51515:   BED
51520:   BRISTOLB
51540:   CHARLOTTESVILLECHAR
51550:   CHESAPEAKECHESAPEAKECHESAPEAKECHESAPEAKECHESAPEAKECHESAPEAKECHESAPEAKE
51570:   COLONIA
51580:   COV
51590:   DANVILLEDANVILLEDANVILLE
```

Linked List:

You are **required** to use a linked list to store the CIS area 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 CIS area 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. (The lecture notes implementation is recommended, as the Carrano implementation has some serious shortcomings.) 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 DCIS 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 CIS area 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 LCIS, and represent that design in a modular structure chart. The structure chart must indicate your design plans for LCIS. 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 CCIS. You must submit this interim design, to the Curator System, no later than midnight Tuesday, April 10. Submit a MS Word.doc file. Do **NOT** compress, (zip), the interim design submission!

Testing:

Obviously, you should be certain that your program produces the output given above when you use the given input files. However, verifying that your program produces correct results on a single test case does not constitute a satisfactory testing regimen. 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.

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 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 `AreaData.txt` and `Actions.txt,` and the corresponding final dump `dbase.txt`, and the corresponding saved database file, named `LCIS.cis.`
- 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 System. 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 shareware program WinZip is very easy to use and is available from the 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 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 `LCIS` must be made by midnight Monday April 24th. There will be **NO late submissions** accepted for `LCIS`. `LCIS` 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 forms 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 `LCIS` 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.