DCIS: Dynamic Census Information System

Fundamental Concepts: separate compilation, elementary operator overloading, dynamic resizable array class of objects

The focus of this programming assignment is to extend your previous work with programmer-defined classes and dynamic memory. This project requires the modification and extension of CCIS. All of the previous functionality of SCIS and CCIS will be retained, unless explicitly removed or modified in this specification. Separate compilation is required for this assignment. Each user-defined class must have its own source code and header files. Other compilation units should be created that reflect the modular decomposition design of the system. Related system sub-sections should be grouped into separate compilation units.

To make the program more memory resource efficient, the dynamically allocated array must be implemented in a class. It must have the ability to dynamically resize itself during execution as the number of area records to be stored grows and shrinks. Additionally, you will add the ability to save the in-memory CIS area database to a file on disk, and to load a CIS area database that was previously saved by your program; see the section on the Dynamic Array Class below for details.

DCIS will normally be invoked from the command-line, and the names of the input files will be specified on the command-line, as:

```
DCIS <InitialCISAreaDataFileName> <DatabaseActionsFileName>
```

DCIS will first load the initial CIS area data file, creating an in-memory database structure, and then read and process actions from the database actions file. As with SCIS and CCIS, when all the specified actions have been processed, DCIS will exit.

File Descriptions:

The initial area population data file and the script actions file will have precisely the same syntax as per SCIS/CCIS, aside from some new actions. Note that use of hard-coded names for these files will annoy the person evaluating your program, and you **will** be charged points for that annoyance.

The format of the CIS area database file, created by the save command, is not specified. Part of your assignment is to design a sensible layout for this file. Note that this means that programs from two different students may certainly have incompatible database file formats, and so will not be able to load database files created by another program. The only restrictions imposed on your database file design are that you should not waste too much space and that the file must be an ASCII text file.

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

```
save <DatabaseFileName>
```

This causes the creation of a CIS area database file on disk, in the current directory. The default extension for the file, "`.cis`" should be automatically appended to the name. As stated above, the format of this file is up to you, subject to light restrictions. Saving does not clear the current in-memory database.

```
load <DatabaseFileName>
```

This causes the reading of the named, (previously saved) CIS area database file, and the creation of a new in-memory database holding the information from that file. The default extension for the file, "`.cis`" should be automatically appended to the name for opening. Any previous in-memory database should be properly deallocated, (but not automatically saved to disk), before the file is read. If the named file does not exist, the following error message must be written to the dump file, "`dbase.txt`" out:

`***Fatal Error***: <DatabaseFileName> does not exist!`

The database filename specified in the load command must be substituted for `<DatabaseFileName>` in the above message. At this point, your program should gracefully shutdown. If the named file does exist, any previous in-memory database should be properly deallocated, (but not automatically saved to disk), before the file is read.

Note that if you do not properly implement the `save` command, there will be absolutely no way to test your implementation of the `load` command. For both input files, a newline character will terminate each input line, including the last. You may assume that all of the input values will be syntactically correct, and that they will be given in the specified order. Updated sample input files for DCIS will be posted on the course website soon. When they are available, an announcement will be posted on the course Web site.

Dynamic Array Class of CIS Area Objects

In this program you are **required** to convert your array database of census area objects into a class. Be aware that a correct implementation will result in no file input or output, (I/O), being performed by any of the array class member functions. (Note: this separation of the I/O from a class also applies to the CIS area class.) The array class will contain constructors, (copy constructor), reporter (get, search, etc.), mutator (set, remove, grow, shrink, sort, etc.) and destructor member function(s). The array class should be viewed and implemented as a container class that is completely unaware of the type of object being stored in it. To this end the CIS area class should overload equality and inequality operators as needed by the array class code.

The Federal Census Bureau has cleverly assigned FIPS numbers so that an ascending FIPS ordering also results in a state/area ordering. Thus the `sort` action command will no longer be used. A **selection** sort on FIPS will be performed after reading the initial CIS area data; all subsequent commands will maintain the ascending FIPS ordering.

Dynamic Array Management

You will initially allocate an array capable of holding exactly 5 CIS area record objects. If the list outgrows the current array size, you will dynamically enlarge the array to hold exactly 5 additional CIS area record objects. If the number of unused array locations grows to 10, you will dynamically shrink the array to hold 5 fewer records (allowing some slack space for future growth).

This crudely mimics the behavior of a C++ vector object. You are specifically forbidden to use any C++ vector objects or any other any STL templates in this program, or to use a linked list of any type in place of the specified array. Violating that restriction would remove one of the major points of this assignment and will certainly result in a major deduction.

Input File Descriptions and Samples:

Initial Area Population File

The format of this file is unchanged from SCIS. A sample `AreaData.txt` input file is shown below.

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

Database Actions File

There is no guaranteed limit on the number of actions. The changes to this file have been discussed previously, (see the File Descriptions section above). A small sample `Actions.txt` input file is shown below.

```
save dcis1
del  51500
add  Alexandria     VA      51510   15.555  111111  44444   55555   16666   33333   17777   4444
del  51590
add  Falls Church   VA      51610   1.999   9999    5000    4444    8333    299     600     244
del  51610
add  Falls Church   VA      51610   1.976   9578    5005    4573    8533    298     604     247
add  Galax          VA      51640   3.778   6670    3663    3007    6219    387     65      41
add  Roanoke        VA      51770   249.914 96397   51807   44590   71907   23395   665     180
add  Winchester     VA      51840   9.000   21111   11111   10000   19999   2111    211     88
del  51840
add  Winchester     VA      51840   9.059   21947   11450   10497   19453   2199    219     86
save dcis2
load dcis1
find 51500
find 51510
find 51840
find 51850
gender      51550
gender      51770
gender      51600
dump
```

Output Description and Sample:

Your program must write its output data to a file named `dbase.txt` — use of any other output file name **will** result in a runtime testing score of zero. Here is a possible output file corresponding to the given sample input files:

```
Programmer:    Dwight Barnette
Dynamic Census Information System
_____________________________________________________________________________________________
Find:        51500      ***MISSING***
Find:            0      Alexandria      VA
Find:        51840      ***MISSING***
Find:        51850      ***MISSING***
Gender:                 Chesapeake      VA      Female%= 51.0   Male%= 49.0
Gender:      51770      ***MISSING***
Gender:      51600      ***MISSING***
_____________________________________________________________________________________________
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
 10> MAX CIS STORAGE
_____________________________________________________________________________________________
```

The first line of your output must include your name only. The second line must include the title "`Dynamic Census Information System`" only. The third line must be a line of underscore characters. The fourth line will contain output from the `Actions.txt` file commands, (find, gender, dump).

The output of dump commands will contain the area data echoed from the current census area database array, aligned under the appropriate headers. The first and last line of each dump must be a line of underscores. The column field headings should be repeated for each display listing resulting from a dump. However, other lines (programmer, program title and underscore lines) are not to be repeated. The `dump` command will be modified slightly. Each area record output will be numbered starting at one. At the end of the CIS area record table listing, the current size of the dynamic array will be output. Note that this is not the same as the number of records stored in the array. Note well that the `Actions.txt` file is not required to end in a dump or save command and multiple dump commands may exist in the file.

You are not required to use the exact horizontal spacing shown in the example above, but your output must satisfy the following requirements:

- You must use the specified header and column labels, and print a row of underscore delimiters before and after the table body, as shown.
- You must arrange your output in neatly aligned columns. Use spaces, not tabs to align your output.
- You must use the same ordering of the columns as shown here, and print the Sqr Miles field with precision three.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in class, in the course notes and on the course Web site about formatting, structure, and commenting your code should be followed. Some specifics:

Documentation:

- You must include the honor pledge in your program header comment, (see below).
- You must include a header comment that describes what your program does and specifying any constraints or assumptions of which a user should be aware, (such as preset file names, value ranges, etc.).
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names suggesting the meaning/purpose of the constant, variable, function, etc.
- Precede every major block of your code with a comment explaining its purpose.
- Precede every function you write with a header comment. This should explain in one sentence what the function does, then describe the logical purpose of each parameter (if any), describe the return value (if any), and state reasonable pre- and post-conditions and invariants.
- Use the assert function to check for error conditions and verify function pre- and post-conditions whenever possible.
- You must use indentation and blank lines to make control structures like loops and if-else statements more readable.

You are also required to conform to the coding requirements specified below.

Coding:

- Implement your solution in a **set of separately compiled source files**, with user-defined header files.
- Use named constants instead of variables where appropriate.
- Use `double` variables for all decimal numbers.
- Implement an array class to store the census area data objects.
- Use C++ `string` objects, not C-style `char` arrays to store character strings, (aside from string literals).
- Declare and make appropriate use of an enumerated type in your program.
- You must make good use of user-defined functions in your design and implementation. To encourage this, the body of `main()` must contain no more than 20 executable statements and the bodies of the other functions you write must each contain no more than 40 executable statements. An executable statement is any statement **other than** a constant or variable declaration, function prototype or comment. Blank lines do not count.
- The definition of `main()` must be the **first function definition** in your source file. You may use file-scoped function prototypes and you may use file-scoped constants. You may also make the `class` declaration statements for your array and census area class types file-scoped (in fact you must do this).
- You may not use file-scoped variables of any kind.
- Function parameters should be passed appropriately. Use pass-by-reference only when the called function needs to modify the parameter. Pass array parameters by constant reference (using `const`) when pass-by-reference is not needed. Pointers should be passed by reference, `const` pointer and/or `const` target as appropriate.

Interim Design:

You will produce an interim design for DCIS, and represent that design in a modular structure chart. The structure chart must indicate your design plans for DCIS. 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 Friday, March 16, (i.e. prior to March 17). 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 modular structure chart 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 `DCIS.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. `DCIS` 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 `DCIS` 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 for your program:

```
//    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.