Linked GIS Linked List and Deep Copy

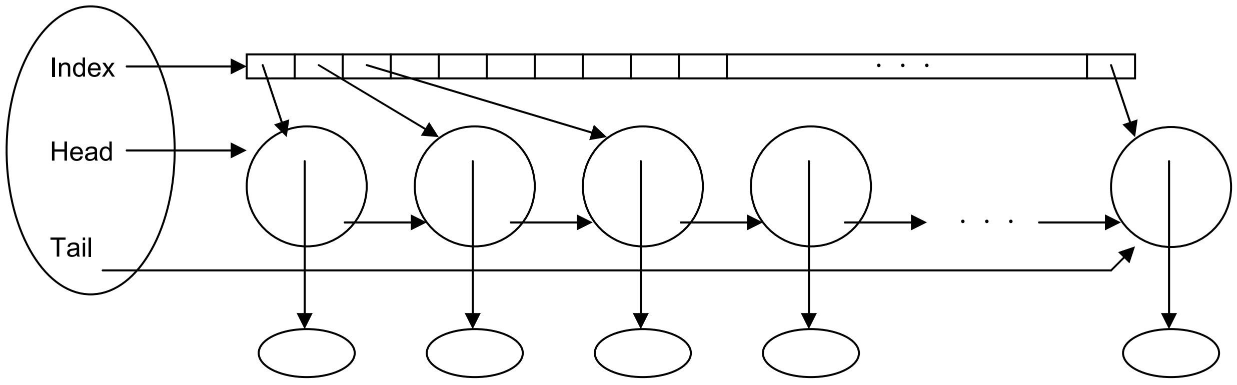
This assignment requires modifying the design and implementation of the Pointy GIS to incorporate a linked list class and additional dynamic memory allocation.

A good design for the Pointy GIS can easily be turned into a good design for this project. Assuming you have a good design for the previous project, you will add a linked list class and modify the designs of several of the other classes from that design

The primary change from the previous project is that the pointers to City objects will now be stored in the nodes of a linked list. It is your choice as to whether to make the list singly- or doubly-linked, but the latter does make some issues easier to handle. The linked list must be implemented as a class, which will make use of a node class as shown in the notes and discussed in the lectures.

Simply using a linked list to organize the city objects would result in inefficient searching. To alleviate that, the linked list class will also contain a dynamically-allocated index array that will store pointers to the linked list nodes. Once the initial city list has been loaded, the linked list will be sorted so that the cities are stored in alphabetic order (as in the previous project); the array will then be allocated and initialized to store pointers to the list nodes. When the linked list performs searches, after the initial loading of data, it will perform a binary search on the array to quickly locate the desired city record.

The linked list object, storing pointers to city records, might look something like this:



There might well be additional data members in the list object itself, and the list might be doubly-linked. But this shows the basic logical structure of the list object. Note well: the GIS <u>contains</u> a list object, the GIS isn't just a list object.

The data structure shown here isn't really optimal. The use of an array parallel to the linked nodes is somewhat strange, but it is a compromise to achieve acceptable search performance without resorting to fancier structures that are beyond the scope of CS 1704.

The final major change is that the neighbor lists will now be allocated dynamically. There will still be a limit of ten neighbors per city, but no limit on the number of city records. Since there is no limit on the number of cities, the list must be able to automatically resize the index array if it is too small. You may use any sensible resizing rules you like, but they must be <u>sensible</u>. Note that making the neighbor list arrays dynamic raises a deep copy issue for the city class; you must provide a proper copy constructor and assignment operator for your city class.

When thinking about your design, it is important to assign each of your classes the correct responsibilities. Give careful thought to the importance of information hiding, especially with respect to the relationship between the controller code and the GIS class. (Yes, that's a strong hint.)

Input Data

There is no change to the input data specification given for the Pointy GIS project.

System Initialization

System initialization proceeds as in the Pointy GIS project, with one change. After reading the list of cities, and storing them in the order they are given in the data file, you will sort the list, using the bubble sort algorithm.

Script File

The script file will have the same format as before.

Log

Everything said about the log output in the previous project specifications still applies. The log file header must echo the names of the input and log files that were used. In addition all searches must be instrumented, by displaying the name and state abbreviation of each city record that is examined during the search.

Since this project is not autograded, the precise formatting of the log file is up to you. In order to make it easier for the TAs to examine your log output, you must also number the commands in your output, starting the numbering at 1.

Implementation Constraints

This specification imposes a number of limitations on how you are allowed to implement your solution. Some of these constraints may appear to be unjustified. They are not. Regardless, failure to conform to them will certainly result in a grade penalty on the project.

- You are forbidden to use anything that is not ISO Standard C++. The course notes present only Standard C++. If you are not sure about something, ask. In particular, you may not use any of the non-Standard abominations commonly presented in the AP Computer Science course. These include, but are not necessarily limited to, `apstring` and `apvector`.
- You must use a linked list class, with an internal array index, as described above.
- You are required to make sure that any memory you allocate dynamically is also deallocated by your program.
- You are forbidden to use any STL containers, except `string`, on this assignment. In view of the restrictions given above, you are thus limited to using statically-allocated arrays for storing lists. This will impose some functional limitations on your solution; that's OK.
- You are required to write classes. You are explicitly forbidden to use `struct` types.
- You must implement your design using separate compilation. In particular, you must have a separate header file and cpp file for each class you write, and at least one cpp file for the driver code that uses those classes.
- With only one exception, all class data members must be private. The exception is that the node class used by the linked list may make its data members public.

This project will not be autograded; instead, you will schedule a demo with a TA. Because of that, some of the restrictions that were imposed on the earlier projects can be dropped. In particular, you may store neighbors in any order you like.

Program Invocation

This program will take the names of the input files and the log file from the command line. Assuming your executable is named `gis.exe`, you would type the following command to run it:

```
gis <city data file name> <route data file name> <script name> <log name>
```

The use of command-line arguments is illustrated in an appendix to the course notes, and will be discussed in class.

Submitting Your Program

Submit all the source files, and the `vcproj` file, for your implementation to the Curator in a single zip archive file. Do not submit unnecessary files! A penalty may be assessed if your zip file is unnecessarily large. You will be allowed to make up to three submissions to the Curator; it will be your choice at your demo as to which will be evaluated.

The *Student Guide* and submission link can be found at: `http://www.cs.vt.edu/curator/`

Evaluation

Shortly before the due date for the project, I will announce which TA will be grading your project. You will schedule a demo with your assigned TA. The procedure for scheduling your demo will be announced later. At the demo, you will perform a build, and run your program on the demo test data, which we will provide to the TAs. The TA will evaluate the correctness of your results. In addition, the TA will evaluate your project for good internal documentation and software engineering practice.

Note that the evaluation of your project will depend substantially on the quality of your design and documentation.

Pledge

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the pledge statement (provided on the course website) in the header comment for your program.

Failure to include the pledge statement may result in a substantial grade penalty.