ECS 40 **Program #5 (35 points, 75 minutes)** **Spring 2015**

Due: Wednesday, May 13th at 11:59pm using handin to p5 directory of cs40a.
New concepts: overloaded operators and linked lists Name of executable: simulator.out
File names: Makefile, main.cpp, city.cpp, city.h, plane.cpp, plane.h, citylist.cpp, citylist.h, authors.txt

For this assignment, you will replace many City methods with corresponding overloaded operators, and replace the Vector class with a CityList class that relies on an underlying linked list. One of our goals is make this conversion as simple as possible. Much of the Vector code will be moved to main.cpp, and we will provide overloaded operators for the CityList to make this transition surprisingly simple.

You may use either your own p4 code, or my p4 code from ~ssdavis/40/p4 (available Thursday morning) as your starting point for the assignment. There is no problem with plagiarism if you use my code. You will find my executable, and the three data files in ~ssdavis/40/p5. As usual, the format and values of your program must match mine.

Further specifications and development guidelines:

1. Creating CityList and CityNode classes. (35 minutes)
   1.1. Your code should compile without warnings after each of these steps. Since you will not creating a CityList yet, the program will always run as it did before. Add CityList.cpp to your Makefile.
   1.2. The CityNode class will contain a City and a CityNode*.
      1.2.1. The CityList class will be a friend of the CityNode class.
      1.2.2. You will need to add a copy constructor to the City class for the CityNode constructor to work properly. Remember to initialize the data members in the order of their declaration.
   1.3. The CityList class will have a CityNode* named head, a CityNode* named tail, and a static int named count.
   1.4. The CityList constructor will simply create an empty list.
   1.5. The CityList destructor will remove all of the CityNodes in the list.
   1.6. Add a static getCount() method. You should call this method without using your CityList object.
   1.7. The += operator will take a City reference as its parameter, and will rely on the tail pointer to append a new CityNode to the end of the list. Note that the code must be aware of when the list is empty.
   1.8. The -= operator will take a City reference as its parameter, find it in the list, and remove the CityNode that contains the City. Note that this will produce a different ordering than the Vector which replaced the City from the end of the array.
      1.8.1. The search will rely on the City::isEqual() method.
   1.9. You will need a const and non-const version of the overloaded [] operator for CityList to allow all of the for-loops in the old Vector code to work. As with arrays, the first CityNode is index zero. You will have to assume that the index provided will always be valid.
2. Moving the Vector code to cities.cpp. (20 minutes)
   2.1. Your code will not compile until you complete all of the following steps.
   2.2. Replace the include of vector.h with that of citylist.h.
   2.3. Remove vector.cpp from your Makefile.
   2.4. Do a global replacement of Vector to CityList in main.cpp
   2.5. Change “cities” from an object to a reference parameter in all functions and function calls in main.cpp. Just walk through the code, line by line. I realize having non-const reference parameters violates our style, but it will make our syntax much simpler.
   2.6. Copy all of the Vector code except the constructor, destructor, and resize() to main.cpp.
      2.6.1. Remove the "Vector::" from the beginning of all the functions, and add a CityList as a reference named “cities” as the first parameter. Be sure to make the CityList a const where appropriate.
      2.6.2. Copy each of the function headers to near the top of main.cpp to serve as prototypes.
   2.7. Do a global replace of "count" with "CityList::getCount()". If you compile now you will find that "++cities.getCount()" is an error in cleanCities() and readCities().
      2.7.1. Rework cleanCities() to utilize the -= operator of CityList.
      2.7.2. You will need to change the code in readCities() to incorporate the CityList += operator by reading the file into a temporary City.
      2.7.3. Both functions should now be much shorter than they were.
   2.8. Now is where all of your hard work writing the CityList class will be rewarded. Do a global replace of "cityArray" with "cities". This will end up compiling because of your overloaded [] operators.

2.9. Your code should now compile without errors and run properly! Unhappily, you will probably have a lot of debugging to do at this step. Take it slow, place breakpoints at each function call in main(), and check the list to see if it makes sense.

  2.9.1. I added some code to the operator[] methods that complained to the screen if an index was out of range.

3. Adding overloaded operators to the City class. (15 minutes)

  3.1. After creating each operator, you will, of course, need to change the calling code. Your code should compile and run perfectly after adding each operator.

  3.2. Change isEqual() to an overloaded operator == of City with a City reference instead of a City pointer, and change the calls to it to calls to the overloaded operator ==. You will have to change the calling parameter to an object instead of pointer.

  3.3. Change readCity() to a typical overloaded extraction operator with an istream reference and a City reference as parameters.

4. General programming standards beyond those provided on the Programming Standards handout.

  4.1. main() may only contain a variable declarations, function calls, and a return statement.

  4.2. All functions must be implemented in .cpp files.

  4.3. Use const wherever possible in function parameters.