For this assignment, you will be supplied with the declaration and partial implementation for a class, whose objects allocate memory dynamically. Your task is to implement the class member functions necessary to provide proper support for deep copy of these objects. Here is the declaration of the class in question:

```
// todoList.h
#ifndef TODOLIST_H
#define TODOLIST_H
#include <iostream>
#include <string>

class todoList {
   friend class Javert;      // class used during testing of your code

public:
   todoList(unsigned int Sz = 10);            // set up empty task list
   todoList(std::string fName);               // create list from file
   bool addTask(const std::string& Task);     // add a task to the list
   std::string getNextTask() const;           // see what the next task is
   bool delTask();                            // remove the next task
   void Display(std::ostream& Out) const;     // see the whole list
   void Save(std::string fName) const;        // save the list to file
   void Clear();                              // remove all the tasks
   todoList(const todoList& Source);          // deep copy support
   todoList& operator=(const todoList& RHS);
   ~todoList();                               // deallocate task list

private:
   unsigned int putNext;                 // index where next task will go
   unsigned int doNext;                  // index of next task to do
   unsigned int Capacity;                // capacity of list
   unsigned int nTasks;                  // number of tasks in the list
   std::string* List;                    // list of tasks
};

#endif
```

The implementations of most of the member functions will be posted on the course website. Your task is to complete the implementation properly.

Aside from correctly supporting deep copy of `todoList` objects, you should take care to make sure that no pointer is ever left storing an inappropriate value if that pointer can later be accessed.

You will submit your work to the Curator, where it will be subjected to rigorous testing to determine whether or not you have provided effective, correct deep copy support. You will submit a file in the following format:

```
// todoList deep copy support
#include <new>
using namespace std;
#include "todoList.h"

// Implementations of deep copy support functions... and nothing else:
```

Note that this file will contain ONLY your implementations of the class member functions that are not supplied. Note also that the organization of the code for `todoList` is a bit different from what you may be used to; in addition to the header file, there are two cpp files, one containing the supplied implementation and one containing only the deep copy support.

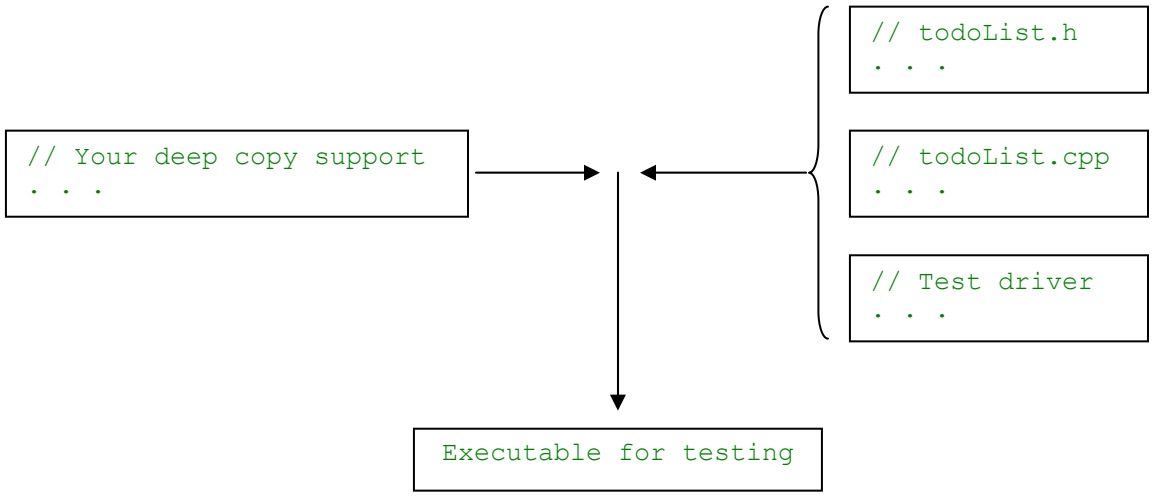
Javert

You have probably noticed that the class header contains a forward declaration and `friend` declaration for a class named `Javert`. This is a class that will be supplied on the Curator when your implementation is tested. Giving `Javert` objects access privileges makes it possible for the test harness to use a `Javert` object to directly examine the results of certain operations, possibly detecting logic errors and sidestepping runtime errors.

You do not have to implement `Javert`, nor will any references to it occur in the code that you write. You are, of course, free to experiment with your own versions of `Javert` if you like, but you must make sure that no references to that occur in the source file you submit to the Curator.

When you submit

What will happen when you submit? The mechanics are simple, but important since if you submit the wrong code then compilation or linking will fail:



The test driver will create instances of `todoList` objects, initialize them in interesting ways, and thoroughly exercise the various ways of making copies of them (which will depend on your implementation).

Evaluation

Do not waste submissions to the Curator to test your program! There is no point in submitting your program until you have verified that it operates correctly. If you waste all of your submissions because you have not tested your program adequately then you will receive a low score on this assignment. You will not be given extra submissions.

Your submitted program will be assigned a score, out of 100, based upon the runtime testing performed by the Curator System. There may or may not be a TA evaluation of your submission.

Submitting Your Program

You will submit this assignment to the Curator System (read the *Student Guide*), and it will be graded automatically. Instructions for submitting, and a description of how the grading is done, are contained in the *Student Guide*.

You will be allowed up to five submissions for this assignment. Use them wisely. Test your program thoroughly before submitting it. Make sure that your program produces correct results for every sample input file posted on the course website. If you do not get a perfect score, analyze the problem carefully and test your fix with the input file returned as part of the Curator e-mail message, before submitting again. The highest score you achieve will be counted.

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

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.