CS 474: Object Oriented Programming Languages and Environments
Spring 2013

Second C++ project

Due time: **7:00 pm on Friday 5/3/2013**

You are required to implement a revised version of the *Painting manager* that you coded previously in C++. This version will require the use of inheritance to model different kinds of paintings, as well as virtual destruction and virtual copying schemes. You are welcome to reuse code that you turned in previously; however, you may have to refactor and extend the code substantiallly in order to comply with the specification below. This project also keeps track of paintings in an art gallery, but now there are three kinds of paintings based on the subject of the painting:

1. *Portrait*—A painting of one or more people.
2. *Landscape*—A painting of a scene in nature.
3. *Still life*—A painting of inanimate objects.

Each painting will have the following items of information.

1. *Title*—This is the title of the painting; it does not need to be unique with respect to other paintings.
2. *Height*—The height of the painting.
3. *Width*—The width of the painting.
4. *id*—A unique integer identifier for the painting. When a painting is created, it is assigned a unique *id* by your system. Subsequently, you refer to each painting by that id.

In addition, portraits will have two additional variables indicating (1) the number of people in the portrait, and (2) a list of names of the people in the painting (if known). Landscapes will have an additional variable indicating the country in which the landscape was taken. Still lifes will add an indication of whether the painting used an oil-based medium or water colors.

Your painting manager must organize its paintings as a collection of linked lists. Each linked list will hold paintings of a given artist (as defined by the artist's first and last name). An additional sequential data structure of your choosing (e.g., a list or an array) will hold a list of artists. For each artist, the list will contain the following three items:

1. *Artist's first name*—The first name of the artist.
2. *Artist's last name*—The first name of the artist.
3. *Artist's paintings*—A pointer to the linked list of paintings by the artist.

You must define at least three C++ classes, namely *LinkedList*, *Painting* (abstract) and *String* (for the paintings' titles, artists' first and last names, names of people in portraits and so on). Abstract class *Painting* will have three concrete subclasses, corresponding to the different kinds of paintings. Linked list nodes will have two data members, a *Painting** pointer that could be bound to instances of any of *Painting*'s concrete subclasses and a pointer to the next node in the list.

Each class will be appropriately equipped with constructors and destructors. The constructors must include at least a programmer-defined default constructor, and a copy constructor. Deep copying of the linked lists is supported through a *virtual copying* scheme. performing a deep copy of the receiver. In addition, you must support

the functionality below. *You must code these classes from scratch. You are not allowed to use predefined libraries, such as STL. The only exception is that you can use the STL Vector class for the list of artists.* Use a command line interface for entering the commands below. Your command line interface will prompt the user for a command, and then execute the command. Here is a list of commands. Make sure not to cause any memory leaks or dangling pointers in the implementation of these commands.

1. a—Add a new artist to your system. The user is prompted for the artist's first and last name. A new, empty linked list of paintings by that artist is also created.

2. p—Add a new painting to the system. The user is prompted for the items of information pertaining to the painting and then an object corresponding to the painting is added to an appropriate list for the painting's artist.

3. r—Removes a painting from the collection. The user is prompted for the painting's id and then the painting is removed from the linked list corresponding to the painting's author. If a painting with the given id does not exist, a message is displayed on the standard output device, and no further action is taken.

4. d—Deletes all paintings by a given artist. The linked list containing the all the painting of a given artists is now empty. Make sure to avoid all memory leaks by using your virtual destruction scheme.

5. l—Lists all the paintings. All painting in the paintings manager are printed by artist's name. The list does not need to be sorted by paintings title or artist's name.

6. c—Clones an artist. The user is prompted for a new first name, then a new artist is created with the same information as an existing artist, except for a different first name. The list of paintings is deep copied using your virtual copy scheme; however, each painting in the new list will receive a new unique id with respect to the existing painting.

7. q—Quits the painting manager.

You must work alone on this project. Save all your code in a collection of header and code files and submit a zip archive with a (short) readme file containing instructions on how to use your Painting Manager. Submit the archive by clicking on the link provided with this assignment. Your code should compile under the GNU C++ compiler. No late submissions will be accepted.