COMP 2404 -- Assignment #3

Due: Friday, March 7, 2014 at 11:00 AM

Goal

You will modify your movie database program from Assignment #2 and add a serialization class in preparation for sending movie information across a network. You will also implement a doubly linked list class to hold the movie information.

Learning Objectives

- begin the adaptation of a repository-style program to a client-server architecture
- learn to format data to send across a network
- gain additional experience with dynamically allocated memory in C++ using linked lists

Instructions:

1. `List` class

a. Implement a doubly linked list class

You will implement a doubly linked list class (called **`List`**) to hold the movies as a collection. You will implement the linked list exactly as we saw in class, with nodes holding pointers to data (i.e. pointers to **`Movie`** objects).
This class should:

- hold pointers to the head and the tail of the list
- provide two add movie functions to add one or more movies to the list
  - one function will take a **`Movie`** pointer as a parameter
  - the other function will take a **`List`** reference as a parameter
- provide a delete movie function to remove movies from the list
  - one function will take a **`List`** reference as a parameter
- manage its memory to avoid memory leaks
- provide any additional utility functions that are required

Notes:
  - All collections in your program must be changed to **`List`** objects, including the master collection of movies
  - Reuse your member functions as much as possible

b. Add an option to the main menu

You will add a new option to the main menu. Option 5 will allow the users to print out the movie list in reverse order. You should traverse the movie list starting at the tail and moving backwards.

2. Serialization

In a future assignment, you will be separating your program into a server process and a client process, where the server process manages the persistent storage for multiple clients, while the client process contains the movie database logic and interacts with the user. However, objects cannot be sent directly between processes. Data that is transmitted and received over a network using sockets must be formatted as an array of bytes. This means that a **`List`** object and the **`Movie`** objects that it contains must be *serialized*, i.e. transformed into a linear format, in order to be sent to another process. Similarly, anything that is received from another process must be *de-serialized* back into a

`List` object and its contents. There are several conventional forms of serialization, including XML, but the simplest is string serialization, which we will use in this assignment.

a. Implement the `Serializer` class

You will add a **`Serializer`** class to your program that serializes **all** communications with the **`Storage`** object. The **`Serializer`** class will contain two public member functions:

- **`void serialize(List& movieList, UpdateType& action, string& serialStr)`** which takes a movie list and a type of action (add, delete, or retrieve) and produces the serialized equivalent
- **`void deserialize(string& serialStr, UpdateType& action, List& movieList)`** which takes a serialized string and extracts the corresponding action type and movie list

Each piece of information in the serialized string will be delimited by an asterisk. The format of the serialized string will be as follows:

- the type of action as a numeric value, followed by an asterisk
- the title, year and numeric genre of the first movie in the list, each followed by an asterisk
- the title, year and numeric genre of the second movie in the list, each followed by an asterisk
- .. and so on until the end of the movie list

For example: a request to add movies “Gravity”, an adventure movie filmed in 2013, and “The Terminator”, an action flick from 1984, will be serialized as:

```
0*Gravity*2013*5*The Terminator*1984*2*
```

where **`0`** is the numeric value for an “add” action, the title and year of the first movie are followed by **`5`** which is the value for the Adventure genre type, followed by the title and year of the second movie and **`2`** for the Action genre.

Note: You may use the C++ standard library **`stringstream`** class to convert to/from string format, or you can use the C library functions (**`sprintf`** and **`sscanf`**). You must **not** use C functions for any other reason! You can find a small example program that uses the C functions here.

b. Modify the `Storage` class

You will add the **`void handleRequest(string& request, string& reply)`** function to your **`Storage`** class. This member function does the following:

- takes a serialized request and de-serializes it into action type and movie list
- calls the existing **`update`** or **`retrieve`** member functions, depending on the action type
- in the case of a retrieve action, **`handleRequest`** serializes the master movie list which it returns as the reply

Note: The **`update`** and the **`retrieve`** member functions do **not** change from Assignment #2, except that they become **`private`** to the **`Storage`** class.

c. Modify the `Server` façade class

You will modify the **`update`** and **`retrieve`** functions of your **`Server`** façade class to support serialization. Both functions must do the following:

- serialize the request to be sent to the **`Storage`** object
- invoke the **`handleRequest`** function on the **`Storage`** object
- the **`retrieve`** function must also de-serialize the **`Storage`** object reply

Note: All communications with the **`Storage`** object must be serialized.

Constraints

- your program must be written in C++, and **not** in C
- do not use any functions from the C standard library (e.g. printf, scanf, fgets, sscanf, malloc, free, string functions)
- do not use any classes or containers from the C++ standard template library (STL)
- do not use low level data types when there is a better one available (e.g. do not use character arrays, use string objects instead)
- do **not** use any global variables or any global functions other than `main`
- do **not** use structs, use classes instead
- objects should always be passed by reference, not by value

Submission

You will submit in *cuLearn*, before the due date and time, the following:

- a UML class diagram (as a PDF file) that corresponds to your program design
- one **`tar`** file that includes:
  - all source and header files for your movie database program
  - a Makefile
  - a readme file that includes:
    - a preamble (program author, purpose, list of source/header/data files)
    - compilation, launching and operating instructions

Grading

- **Marking breakdown:**

Component	Marks
UML class diagram	10
List class	30
Serializer class	20
Modifications to Server	25
Modifications to Storage	15

- **Deductions:**
  - Up to 50 marks for any of the following:
    - the code does not compile using g++ in the VM provided for the course
    - the code cannot be tested because the program consistently crashes
    - the design does not generally follow correct design principles (e.g. data abstraction)
    - prohibited library classes or functions are used
  - Up to 20 marks for any of the following:
    - the Makefile or readme file is missing
    - global variables, global functions, or structs are used
    - objects are passed by value
    - the code is not correctly separated into header and source files
  - Up to 10 marks for missing comments or other bad style (non-standard indentation, etc.)
- **Bonus marks:**
  - Up to 5 extra marks are available for fun and creative additional features