CSCI 2270 – Data Structures and Algorithms
Instructor: Hoenigman
Assignment 6
Due Friday, March 6 by 3pm

Binary Search Trees – Part Two

An online movie service needs help keeping track of their stock. You should help them by developing a program that stores the movies in a Binary Search Tree (BST) ordered by movie title. For each of the movies in the store's inventory, the following information is kept:

- IMDB ranking
- Title
- Year released
- Quantity in stock

Your program will have a menu similar to assignments 3 and 4 from which the user could select options. In this assignment, your menu needs to include options for finding a movie, renting a movie, printing the inventory, and quitting the program.

Your program needs to incorporate the following functionality from assignment 5. Use the same Assignment5Movies.txt file that you used for assignment 5 and pass the name of the file as a command line argument.

1. **Insert all the movies in the tree**.
   When the user starts the program they will pass it the name of the text file that contains all movie information. Your program needs to handle that command line argument, open the file, and read all movie data in the file. From this data, build the BST ordered by movie title. All other information about the movie should also be included in the node in the tree. *Note: the data should be added to the tree in the order it is read in.*
2. **Find a movie.**
   When the user selects this option from the menu, they should be prompted for the name of the movie. You program should then search the tree and display all information for that movie. If the movie is not found in the tree, your program should display, "Movie not found."
3. **Rent a movie.**
   When the user selects this option from the menu, they should be prompted for the name of the movie. If the movie is found in the tree, your program should update the Quantity in stock property of the movie and display the new information about the movie. If the movie is not found, your program should display, "Movie not found." If the movie is found in the tree, but the Quantity is zero, display "Movie out of stock.".
4. **Print the entire inventory.**

When the user selects this option from the menu, your program should display all movie titles and the quantity available in sorted order by title. See the lecture notes on in-order tree traversal for more information.

5. **Quit the program.**
When the user selects this option, your program should generate the Assignment6Output.txt file and exit.

Additional Functionality for Assignment 6

1. **Delete a movie.**
When the user selects this option, they should be prompted for the title of the movie to delete. Your code should then search the tree for that movie, delete it if it's found, re-assign to the parent and child pointers to bypass the deleted node, and free the memory assigned to the node. If the movie is not found in the search process, print "Movie not found" and do not attempt to delete.

   A movie node should also be deleted when the quantity goes to 0 for any movie.

2. **Count movies in the tree.**
When the user selects this option, your program should do an inorder tree traversal and count the total movie nodes in the tree, print the value, and add the information to the json object.

3. **Delete the tree in the destructor using a postorder traversal.**
When the user selects quit, the destructor for the MovieTree class should be called and in the destructor, all of the nodes in the tree should be deleted. You need to use a postorder tree traversal or you will get segmentation fault errors.
4. **Generate a json file.**
For each of the add movie, delete movie, traverse tree, count nodes, and rent movie operations performed, your program should add a json object to the json output for the program, and then write that output to a file when the user quits the program. **The file should be called Assignment6Output.txt.**

   The add movie operation is used for each movie node added to the movie tree when you are building the tree. The delete movie operation is used when the user deletes a movie or when a movie is deleted because the quantity is 0. The traverse tree operation is used when the selects Print the entire inventory from the menu. The count nodes operation is used when the user selects Count movies in the tree from the menu. The rent operation is used each time a user selects rent movie and the movie is successfully rented. The format for the json for each operation type is shown in Appendix A.

Implementation details
Your BST should be implemented in a class. You are provided with a MovieTree.h file on Moodle that includes the class prototype for this assignment. You need to implement the class functionality in a corresponding MovieTree.cpp file and Assignment6.cpp file. To submit your work, zip all files together and submit them to COG. If you do not get your assignment working on COG, you will have the option of a grading interview.

Appendix A – json key:value pairs that COG expects

When your program performs an operation, such as deleting a node, or adding a node to the tree, you should add information specific to that operation to your json object that will eventually be written to a file just before the program quits. Each operation performed should be assigned the operations counter as the key, and then depending on the type of operation, other information is included. The operation object should contain a key called “operation”, which can have the values “add”, “delete”, “traverse”, “rent”, or “count”. A sample for each operation type is shown here:

```
{
  "1":{
    "operation":"add",
    "parameter":"movie title",
    "output":[
      "movie title 1",
      "movie title 2",
      "movie title 3",
      "movie title z"
    ]
  },

  "2":{
    "operation":"delete",
    "parameter":"movie title",
    "output":[
      "movie title 1",
      "movie title 2",
      "movie title 3",
      "movie title z"
    ]
  },

  "3":{
    "operation":"traverse",
    "output":[
      "movie title 1",
      "movie title 2",
      "movie title 3",
      "movie title z"
    ]
  },
  "4":{
    "operation":"count",
    "output":50
  },
  "5":{
    "operation":"rent",
    "parameter":"movie title",
    "output":10
  }
}
```

In this example, the first operation performed is an “add”. The value for “parameter” is the movie title of the added movie node, and the “output” value is the array of movie titles observed as we traverse the tree searching for the correct spot to add the movie node. The operation has a “1” because we did it first. If we perform 50 add operations, each of them would be given a number 1…50 in the order they were performed.

The “delete” operation includes the same data as the “add” operation. The “traverse” operation includes the key for the operation, and “output” is the titles observed for each movie node in the traversal. The “count” operation only has “output”, which is the integer count of the number of movie nodes in the tree. In the “rent” operation, the “parameter” value is the title of the movie rented, and the “output” is the quantity of that movie remaining after the rental.

For example, when you are building the tree from the data in the text file, the first movie you add is “Shawshank Redemption”. You would add an object to your json, such as:

```
“1”:{
     "operation":"add",
     "parameter":"Shawshank Redemption",
     "output":[
               "Shawshank Redemption",
     ]
  },
```

The next movie you add is The Godfather. You would add another object to your json that looked like:

```
"2":{
     "operation":"add",
     "parameter":"The Godfather",
     "output":[
               "Shawshank Redemption",
               "The Godfather",
     ]
},
```

The process continues for each movie added, with the operations counter increasing each time. When the user selects an option from the menu, you maintain the same counter, increment the counter for the next object added to the json object, and add to the same json object. For example, if the user selects print the tree after building the tree, you would add an object to your json that looked like:

```
"51":{
     "operation":"traverse",
     "output":[<array of strings of all movies titles in
     inorder tree traversal>]
},
```