```cpp
/*
ALEX FRIEDBERG

tree .cpp
*/

#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include "tree.h"

using namespace std;

tree::tree() {
    root = NULL;
}

void tree::displayInOrder(TreeEntry *entry) {
    if (entry != NULL)
    {
        displayInOrder(entry->left);

        entry->data->printShow();
        cout << endl;

        displayInOrder(entry->right);
    }
}

void tree::displayAllTitles(TreeEntry *entry) {
    if (entry != NULL)
    {
        displayAllTitles(entry->left);

        cout << entry->data->getName() << endl;

        displayAllTitles(entry->right);
    }
}

void tree::displayAll() {
    displayInOrder(root);
}

void tree::displayAllTitles() {
    displayAllTitles(root);
}
```

```
50
51  void tree::addToTree(TREE_DATA_TYPE *newData)
52  {
53      TreeEntry *newPtr = new TreeEntry;
54
55      // Add new data in the new node's data field
56      newPtr->data = newData;
57      newPtr->left = NULL;
58      newPtr->right = NULL;
59
60      //  If the BST is empty, insert the new data in root
61      if (root == NULL)
62      {
63          root = newPtr;
64      }
65      else   //  Look for the insertion location
66      {
67          TreeEntry   *treePtr = root;
68          TreeEntry   *targetNodePtr;
69
70          while (treePtr != NULL)
71          {
72              targetNodePtr = treePtr;
73              if(newData->isSameShow(treePtr->data))
74                  // Found same data; ignore it.
75                  return;
76              else if (newData->getStartYear() < treePtr->data->getStartYear())
77                  // Search left subtree for insertion location
78                  treePtr = treePtr->left;
79              else   //  newData > treePtr->data
80                     // Search right subtree for insertion location
81                  treePtr = treePtr->right;
82          }
83
84          //  "targetNodePtr" is the pointer to the
85          //  parent of the new node.  Decide where
86          //  it will be inserted.
87          if (newData->getStartYear() < targetNodePtr->data->getStartYear())
88              targetNodePtr->left = newPtr;
89          else  // insert it as its right child
90              targetNodePtr->right = newPtr;
91      }
92  }
93
94  void tree::displayActorsByShowName(string showName) {
95      cout << "DISPLAYING ACTORS BY SHOW NAME: " << showName << endl;
96      displayActorsByShowName(root, showName);
97      cout << endl;
98  }
```

```
 99
100 void tree::displayActorsByShowName(TreeEntry *entry, string showName) {
101     //PRIVATE
102     if (entry != NULL)
103     {
104         if (entry->data->getName().compare(showName) == 0) {
105             entry->data->printActors();
106         }
107
108         displayActorsByShowName(entry->left, showName);
109         displayActorsByShowName(entry->right, showName);
110     }
111 }
112
113 void tree::displayShowsByActorName(string actorName) {
114     cout << "DISPLAY ALL SHOWS BY ACTOR NAME: " << actorName << endl;
115     displayShowsByActorName(root, actorName);
116     cout << endl;
117 }
118
119 void tree::displayShowsByActorName(TreeEntry *entry, string actorName) {
120     //PRIVATE
121     if (entry != NULL)
122     {
123         if (entry->data->containsActorName(actorName)) {
124             entry->data->printShow();
125         }
126
127         displayShowsByActorName(entry->left, actorName);
128         displayShowsByActorName(entry->right, actorName);
129     }
130 }
131
132 void tree::displayShowsByDateRange(int yearRangeStart, int yearRangeEnd){
133     //Assumes rubric means that the show started within the date range given
134     cout << "DISPLAY ALL SHOWS REALEASED BETWEEN: " << yearRangeStart << " and
        " << yearRangeEnd << "." << endl;
135     displayShowsByDateRange(root, yearRangeStart, yearRangeEnd);
136     cout << endl;
137 }
138
139 void tree::displayShowsByDateRange(TreeEntry *entry, int yearRangeStart, int
      yearRangeEnd) {
140     //PRIVATE
141     if (entry != NULL)
142     {
143         if (entry->data->getStartYear() >= yearRangeStart && entry->data-
            >getStartYear() <= yearRangeEnd) {
144             entry->data->printShow();
```

```
145         }
146
147         displayShowsByDateRange(entry->left, yearRangeStart, yearRangeEnd);
148         displayShowsByDateRange(entry->right, yearRangeStart, yearRangeEnd);
149     }
150 }
```