```
/* BSTv3.cpp
 *
 * Binary Search Tree Implementation.
 */

#include "BST.h"

#include <fstream> //to import file

void BinarySearchTree::readFile()
{
  string placeHolder; // used for file read in problems
  string showName,tempShowName; // TV show name strings
  int releaseDate; // year of release
  // same as in interface file
  int endDate;
  string genre;
  string url;

  string actorName; // actor names for each movie
  fstream inFile; // fstream object to read in file

  //deleted a repeat entry for babylon 5 in the txt file
    // and removed some extra lines of space
  inFile.open( "TVshowBST.txt" ); // opens selected file

  while( inFile.is_open() ) // only operates when file is open
    {
      //check:      cout << "inFile obj is open" << endl;

      LinkedList L1; // list object that stores actor names for each TV
show
      tempShowName = showName;
      getline( inFile,showName,'('  ); //read data from inFile obj until
                                         // '(' char, put in showName

      if( showName == tempShowName ) //checks if at end of TV shows
        {
          cout << "File has been read. \n" << endl;
          inFile.close();            //closes current file
          break;                     //breaks out of loop
        }

      //check:      cout << showName << endl;
      inFile >> releaseDate;
      //check:      cout << releaseDate << endl;
      inFile >> endDate;
      endDate = endDate * -1; //to negate reading
                                // the dash from file as a negative
      //check:      cout << endDate << endl;

      inFile >> placeHolder;
      inFile >> genre;
      //check:      cout << genre << endl;

      inFile >> url;
      //check:      cout << url << endl;
      getline( inFile,actorName ); //to get rid of space

      do{                          //gets all actor names
```

```
        getline( inFile,actorName );
      //check:              cout << actorName << endl;
        L1.AddNodeToEnd( actorName ); //add actor name to list
      }while( actorName.length() > 0 ); // while actor name not empty
      //check:            cout << "Actors in the List: \n" << endl;
      //check:            L1.PrintNodes();

      // passes read in info to function
      AddNode( releaseDate,showName,endDate,genre,url,L1 );

      //check:            cout << "movie added" << endl;
    }
}


// AddNode()
// Add (insert) new item into the BST, whose
// root node is pointed to by "rootPtr". If
// the data already exists, it is ignored.
void BinarySearchTree::AddNode( int newStartDate2,
                                string showName2,
                                int endDate2,
                                string genre2,
                                string url2,
                                LinkedList L12 )
{
  TreePtr newPtr;
  newPtr = new BSTreeNode;

  // Add new data in the new nodeâ€™s data field
  newPtr -> startDate = newStartDate2;
  newPtr -> showName = showName2;
  newPtr -> endDate = endDate2;
  newPtr -> genre = genre2;
  newPtr -> url = url2;

  newPtr -> L = L12;
  //check if copies over correctly:  newPtr -> L.PrintNodes();

  newPtr -> leftPtr = NULL;
  newPtr -> rightPtr = NULL;

  // If the BST is empty, insert the new data in root
  if( rootPtr == NULL )
    {
      rootPtr = newPtr;
    }
  else // Look for the insertion location
    {
      TreePtr treePtr = rootPtr;
      TreePtr targetNodePtr;

      while( treePtr != NULL )
        {
          targetNodePtr = treePtr;
          if( showName2 == treePtr -> showName )
            // Found same data; ignore it.
            return;
        // insert new node alphabetically
          else if( showName2 < treePtr -> showName )
```

```
            // Search left subtree for insertion location
            treePtr = treePtr -> leftPtr;
          else // showName2 > treePtr -> showName
            // Search right subtree for insertion location
            treePtr = treePtr -> rightPtr;
        }
      // "targetNodePtr" is the pointer to the
      // parent of the new node. Decide where
      // it will be inserted.
      if( showName2 < targetNodePtr -> showName )
        {
          targetNodePtr -> leftPtr = newPtr;
        }
      else // insert it as its right child
        {
          targetNodePtr -> rightPtr = newPtr;
        }
    }
}

// passes root pointer to the private version of this function
void BinarySearchTree::SearchNode( string newShowName ) // public
{
  SearchNodeInBST( rootPtr, newShowName );
}

// SearchNodeInBST()
// Find a given node by "key" in BST. If successful, it
// returns the pointer that points to the node with "key";
// otherwise, it returns NULL. It uses preorder traversal.
void BinarySearchTree::SearchNodeInBST(TreePtr treePtr, string
newShowName)//priv
{
  /*check:
    cout << "Searching for " << newShowName
    << " but " << treePtr
    -> showName << " is the current showName."
    << endl;
  */

  if( treePtr != NULL )
    {
      //check:
      // cout << "Searching for " << newShowName
      //      << " currently looking at " << treePtr -> showName << endl;

      if( newShowName == treePtr -> showName )
        {
        // prints out list of actors for this TV show
          cout << "\nFound: " << treePtr -> showName << endl;
          cout << "Actors of " << treePtr -> showName << ": " << endl;
          treePtr -> L.PrintNodes();
        return;
        }
      else if( newShowName < treePtr -> showName )//search preOrder
        // Search for "key" in left subtree
        SearchNodeInBST( treePtr -> leftPtr, newShowName );
      else // (key > tree_ptr -> showName)
        // Search for "key" in right subtree
        SearchNodeInBST( treePtr -> rightPtr, newShowName );
    }
  else {
    // TV show not in database
    cout << "Didn't find: " << newShowName << endl;
    return;
  }
}

// passes root pointer to the private version of this function
void BinarySearchTree::ActorSearchNode( string newActorName ) // public
{
  ActorSearchNodeInBST( rootPtr, newActorName );
}

// ActorSearchNodeInBST()
// same as SearchNodeInSBT(), but searches by actor name
// prints out each show an actor is in
void BinarySearchTree::ActorSearchNodeInBST( TreePtr treePtr,
                                             string newActorName )//priv
{
  if( treePtr != NULL)
    {
      // Print left BST subtree
      ActorSearchNodeInBST( treePtr -> leftPtr, newActorName );
      // Print showName if actor is in show
      if( treePtr -> L.SearchList( newActorName ) == true )
        {
          cout << "Actor " << newActorName << " is in "
                                           << treePtr -> showName <<
endl;
        }
      // Print right BST subtree
      ActorSearchNodeInBST( treePtr -> rightPtr, newActorName );
    }
}

// passes root pointer to the private version of this function
void BinarySearchTree::DecadeSearchNode( int releaseDate,
                             int stopDate ) // public
{
  DecadeSearchNodeInBST( rootPtr, releaseDate, stopDate );
}
// DecadeSearchNodeInBST()
// same as SearchNodeInSBT(), but searches by decade of release
// prints shows by decade TV show released in
void
BinarySearchTree::DecadeSearchNodeInBST( TreePtr treePtr,
                             int releaseDate,
                             int stopDate )//priv
{
  if( treePtr != NULL)
    {
      // Print left BST subtree
      DecadeSearchNodeInBST( treePtr -> leftPtr, releaseDate, stopDate );
      // Print showName if released within that decade
      if( (treePtr -> startDate >= releaseDate) &&
          (treePtr -> startDate <= stopDate) )
        {
          cout << "Show: " << treePtr -> showName << " is release between
"
```

```
            << releaseDate << " and " << stopDate  << endl;
        }
      // Print right BST subtree
      DecadeSearchNodeInBST( treePtr -> rightPtr, releaseDate, stopDate
);
    }
}

// PrintInOrder()
// Print BST using InOrder traversal
// also prints out all TV show names
void BinarySearchTree::PrintInOrder() //public
{
  PrintBST_InOrder( rootPtr );
}
void BinarySearchTree::PrintBST_InOrder(
                                        TreePtr treePtr ) //priv
{
  if( treePtr != NULL)
    {
      // Print left BST subtree
      PrintBST_InOrder( treePtr -> leftPtr );
      // Print Root node data
      cout << treePtr -> showName << endl; //displays showName
      // Print right BST subtree
      PrintBST_InOrder( treePtr -> rightPtr );
    }
}
```