Seth Cram

CS121

listT.h and listT.cpp

Create a basic linked list class. Within that class there needs to be a private node struct that has string actorName, so we can store the actors/actresses of each show later on. From that struct declare a start and end node pointer, so we can keep tabs on our doubly linked list. Now for the public functions.

Declare a boolean type function named SearchList() that takes a string as an argument. The actor/actress name we are looking for is passed into this function. In the body of SearchList(), use a while loop to traverse the list until the current node pointer is NULL. That will walk down the list to the end. In the body of that while loop, at each node check to see if the node’s actorName is the same as the passed in string. If it is, return true. If you get to the end of the list without finding the string, return a false.

Now for a void type AddNodeToEnd() function that takes a string as the argument. The actor/actress name will also be passed into this function. Make sure to create a new node pointer, with its next set as NULL initially. Account for if the list is empty, then if it is empty just add the node on, but if it isn’t empty just add the node onto the end of the list using the end pointer.

Finally, the void type PrintNodes() function won’t take anything in its argument. Use a while loop to traverse the list as used in the SeachList() function but instead of comparing the actor/actress names in the body, just print out the current node’s actor/actress name.

BST.h and BSTv3.cpp

Create a Binary Search Tree class, and within that private class declare string showName, int startDate, int endDate, string genre, string url, and LinkedList L to store all of the information pertaining to each individual TV show that we will later read in. Also within the class, declare a left pointer and right pointer to structure the BST (Binary Search Tree) later on. Now create some public functions.

Void readFile() has no argument and will read in all the information from TVshowBST.txt, calling the AddNode function to pass that information into the BST. Redeclare all the variables declared in the BST class, besides the tree pointers. Also declare string placeHolder, string tempShowName, string actorName, and fstream inFile. Use the fstream library’s open function with our fstream object, inFile, to open TXshowBST.txt. Now create a while loop that only loops when the file is open. Within the body of that loop, create a LinkedList object L1. Then check if, after setting tempShowName to showName and setting showName equal to some information read in from the txt file, tempShowname and showName are still the same. If they are, close the file and break out of the loop. Otherwise, go on to read in all the information enclosed in the txt file with the above declared variables. For the actorName’s create a do while loop that only loops when there is an actorName being read in. Within that loop, read an actorName in from the txt file and then use the AddNodeToEnd() function with the actorName as an argument to input each show’s actorName’s to its own linked list. Finally, before the end of the while loop, add input to the AddNode() function that we will define later on. As its argument, pass in the releaseDate, showName, endDate, genre, url, and L1. This function will create and add a node onto our tree with all the information of a single TV show.

So this AddNode() function is void, and takes an int newStartDate2, string showName2, int endDate2, string genre2, string url2, and LinkedList L12 as an argument. In the body of the function, first create a new BST node and give this new node all the passed in information. Set the new node’s left and right pointers to NULL, and then insert the new node into the BST. To insert it, if the root pointer is NULL, just set our new node equal to the root pointer. This adds it onto the tree as the key node. If it isn’t, create a while loop that traverses the tree until the tree pointer is NULL (it reaches the end of the tree). In the body of this while loop, recursively insert the node into the left subtree its showName is closer to the beginning of the alphabet than the key, or insert it into the right subtree if it’s closer or equal to end of the alphabet than the key’s showName. Now it’s on to the search and print functions.

Create a public void SearchNode() function that takes string newShowName as the argument. This function passes both the root tree pointer and newShowName into SearchNodeinBST(). SearchNodeinBST() is a private void type function that has a tree pointer named treePtr and string newShowName passed in as the argument. Within this function, use a PreOrder search method to search for a specific showName in the database. If it’s found, print out the list of actorNames using the PrintNodes() function for that node. To search for the showName, recursively look through the left subtree, then recursively look through the right subtree. At each node, check if the show name is the show name we are looking for before recursing again. Make sure to return if the show is found, and return if the show isn’t found.

Now make a public function called void ActorSearchNode that takes string newActorName as the argument. This function passes both the root tree pointer and newActorName into ActorSearchNodeinBST(). ActorSearchNodeinBST() is a private void function that searches the BST by actor name and prints out each show the actor is in. Search the tree just as we did above in the SearchNodeinBST() function. Except this time, use an InOrder search to look through the entire BST from a-z, and print out each showName that the SearchList() function turns up as true when the newActorName is passed in. Before checking, make sure to go through the left subtree recursively, and then after checking, go through the right subtree. This allows for an alphabetic search.

For our last search function, declare void DecadeSearchNode() as a public function that takes both int releaseDate, and int stopDate, as the argument of the function. This function passes the root tree pointer and both integers into DecadeSearchNodeinBST(). DecadeSearchNodeinBST() is a private function that searches in the same fashion that ActorSearchNodeinBST() does. But this function’s check for printing is if the node’s current startDate is equal to or bigger than the releaseDate and smaller than or equal to the stopDate. It’ll print out the showName if it was released in between the specified years.

Lastly, the PrintInOrder function is public and a void function that has no argument. It passes the rooPtr into the private PrintBST\_InOrder() function. PrintBST\_InOrder() is void as well and takes a tree pointer for the argument. It searches through the tree the same way DecadeSearchNodeInBST() and ActorSearchNodeInBST() do, but there’s no check so it just prints out every tree node’s showName.

TestBST.cpp

In the created main function declare a BinarySearchTree object T1. Use T1 to statically call all the following functions. Use it to call readFile() to read in the txt Tv show database file. Then with the PrintInOrder() function, make sure it displays all the show titles. Then use the searchNode() function to look for any desired show title, but make sure to include a space after the desired title. Do the same for ActorSearchNode() without a space at the end and for the wanted actor. Next, finish with DecadeSearchNode(), including two years to display what shows were released between those two dates.

General:

Make sure that listT.h uses namespace std and includes iostream so all the other functions also inherit them. Link TestBST.cpp to BSTv3.cpp, BSTv3.cpp to BST.h, BST.h to listT.cpp, and finally listT.cpp to listT.h by including the file name at the top of the linked file. Also make sure to include fstream at the top of BSTv3.cpp. For all of the functions, make sure to make them members of the BinarySearchTree or LinkedList class in order to give them access to their private variables.