**Name:** Patrick Tate

**ID:** 108350446

**Date:** 05/02/2018

**Program:** CSCI 2421 Final Project: Actor/Movie Database Program

**2. Program Description**

This program reads in data from two csv files, actor-actress.csv and pictures.csv. Each csv file is placed into a binary search tree sorted by name, ie either the name of the actor/actress or name of the picture. The user will have the chance of entering the actor/actress database or the picture database. Once they have selected their database, they will be presented with a menu of options, which will include adding a record, search for a record and modify the fields, search for a record and delete the fields, sort the database by any other sortable field, do a complete search for any searchable field, a partial search for any searchable field, and also printing the updated database to a new csv file.

**3. Overall Software Architecture**

The main data structure used is a binary search tree, one for actor-actress.csv and one for pictures.csv, both sorted by name. Each bst serves as the database for either actors/actresses or pictures. The user experience involves navigating through various menus, choosing which database to enter, and which actions they wish to take within that database.

Most functionality involves searching the BST and deleting nodes/adding nodes correctly within a BST.

\*\*\***Activity Diagram at bottom of document\*\*\***

**4. Input Requirements**

Obviously the two main inputs are the actor-actress.csv and pictures.csv, each of which is read into a binary search tree. Since the files are csv, the data is read in using a getline function and string stream to convert any non string variables into their appropriate values. For the pictures database, name, genre 1, genre 2, release, and synopsis are strings, while year, nominations, rating, duration, and metacritic are ints or doubles, which are then converted using string stream. For the actor/actress database, award, name, and film are strings, while year is an int and winner is a bool, both are converted using sting stream.

Some fields have empty data, eg just a blank space between commas, this affects searches since there is no value to search for in that field, or sorting, eg a Node with no meta critic has its score set to 0, etc.

These data types are kept in mind when the user is manually entering info to change/add a record, to prevent them from entering garbage or incorrect values. For the actor/actress database, award, name, and film can all be obtained with a getline, winner is input with an int, either a 1 or 0, for easy range testing and conversion to bool value, year is also read in as an int. For the pictures database, name, genre 1, genre 2, release, and synopsis are all read in as strings using getline, nominations, duration, and metacritic will all be ints, obviously a range of 0-100 will be used for the metacritic value, rating will be a double and user will be prompted to enter a rating value from 0.0 to 10.0.

**5. Output Requirements**

Once a record is found it, it will be displayed to the user. Printing is done in order, in alphabetical order. There are different print functions defined for trees that are sorted by int, double, and string. The default database is keyed on string so the default print database uses a print in order string function.

Another output is sending the updated database to a csv file. Since the data is already organized and updated in the bst, an info line is sent first “name,year,awards,etc” followed by the database info in order. The function appends the data, so if this function is called twice, the second call will append data.

**6. Problem Solution Discussion**

The main functionality of this program involves utilizing a binary search tree to store data, search for data, add data, modify data, print data, and send data to a new file. A separate functions.h file contains a function that loads a bst with data, and a function that takes in a bst and sends that data to a new csv file. Most of the program’s functionality involves functions within the binary search tree class, or within the specific database.

First the user is confronted with a menu asking them which database they want to enter, once they have chosen a database they will be presented with another menu asking them if they want to add a record, modify fields, delete fields, sort by any other sortable field, do a complete search, do a partial search, print the database, sort and print, or return to previous menu, all of which are functionality of the binary search tree and will mainly utilize functions from that class.

**7. Data Structures**

A binary search tree is required for the databases, but it makes sense to use a bst because lots of functionality will involve continually searching the database, and since the time complexity of searching a bst is O(log(n)), and our database will have around 10,000 records.

The function that does a partial/complete search stores any found records in a vector of actor or picture structs. A vector does have a much slower insertion (O(n)) and search (O(n)) than a bst, but I’m operating under the assumption that the results of a partial search won’t be more than a couple hundred records at most, which would make inserting, searching, and accessing the vector trivial, and the ease of using a vector is always appreciated. Since a vector has a built in size function, makes it easy to tell the user “your partial search yielded 200 results.” If the size of the vector/results from the search are greater than 5, the user has the option to print all the results or re-search the vector.

Two structs, one for actor/actress database, and one for pictures database are used. It makes sense to use two different structs because the files don’t have the same fields, and it would be wasteful and confusing to try and use just one struct.

Resorting the database involves traversing the entire tree in order and placing that Node into a new binary search tree keyed on a different value. Note this doesn’t change what the main database binary search tree is keyed on, it just creates a new binary search tree and prints it in order.

**8. User interface scheme**

Initial menu has 3 options, 1 enter picture database, 2 enter actor database, 3 exit program. Once a database is selected, each database will then have another menu where the user can choose what actions to take, 1 add a record, 2 search for an actor/picture by name and modify the fields, 3 search for an actor/picture and delete the record, 4 sort and print the database by any sortable field, 5 do a complete search on any searchable field, 6 do a partial search on any searchable field, 7 write the updated database to a new csv file, 8 print the updated database, 9 return to main menu, and 10 exit program. The options of sorting by any sortable field, and doing a partial search will most likely result in another menu of options, asking the use which field they want to sort by or what they want to do with the x number of results from their partial search.

**9. Status of Application**

This program compiles and runs on the use grid. This program was developed on macOS using Xcode.

**Activity Diagram Note:** since the main functionality will be navigating through nested menus, it didn’t make sense to me to have a clear end point in the diagram, since exiting the program is dependent on when the user chooses to.