1.

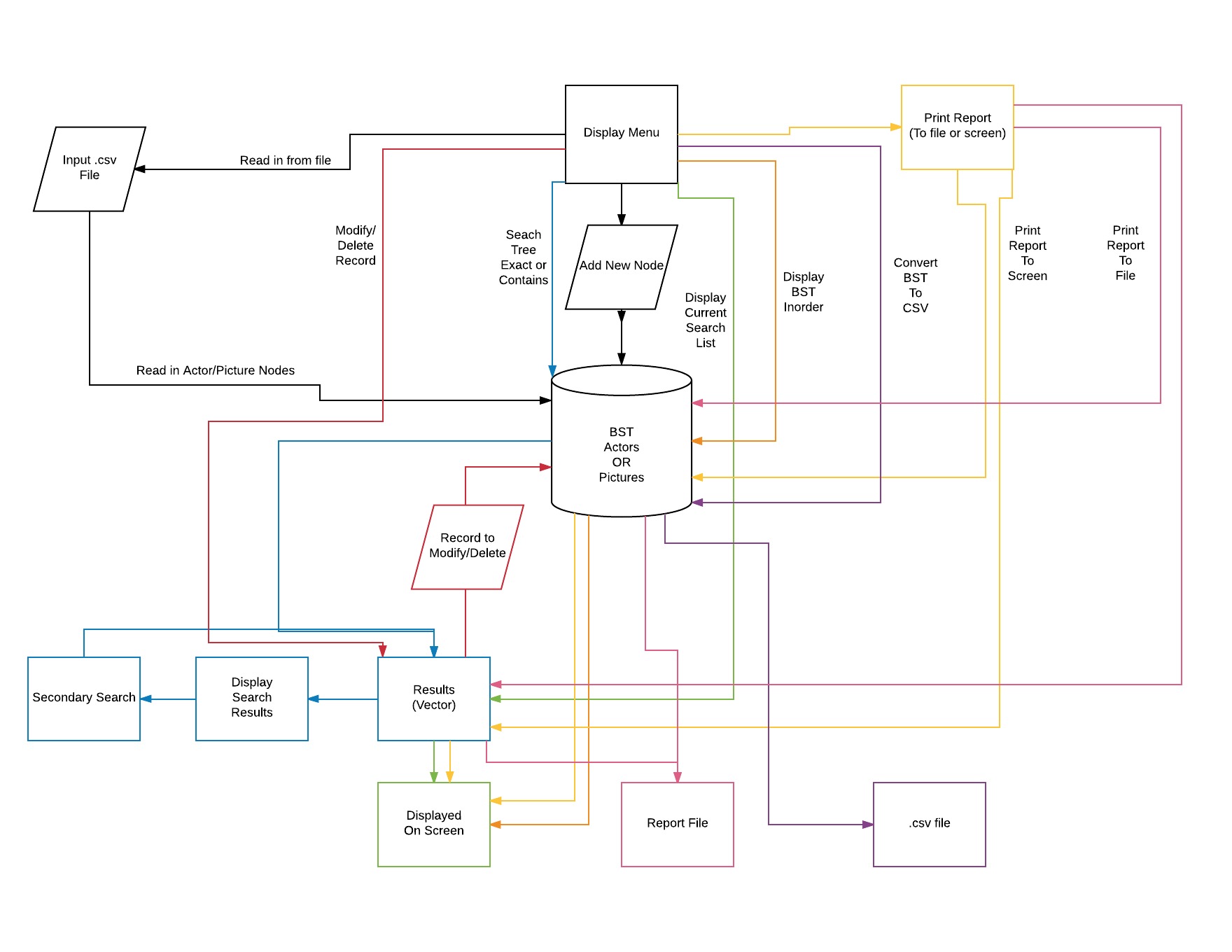
Binary Search Tree Database

Matt Sullivan

SID: 104396771

12/1/17

1. This program creates a database of Academy Awards winners, stores them in one of two binary search trees (one for actors/actresses, the other for movies). After the binary search trees are read in from .csv files, the user is able to add, modify, or delete entries, as well as search the database using strings and sort the database using any field. The user will also be able to select fields to print, print a report, or export the data into a file.
2. Due to the limited size, and same basic structure of the two BSTs, I made one program flow. It can be applied to either tree/vector as they have the same functionality.



4. External inputs include: Ability to add information to their respective BST, both individual actors/actresses, in the format of Year,Award,Winner,Name,Film (int, string, int, string, string) or pictures: name,year,nominations,rating,duration,genre1,genre2,release,metacritic,synopsis  
(string, year, int, float, int, string, string, string, int, string)

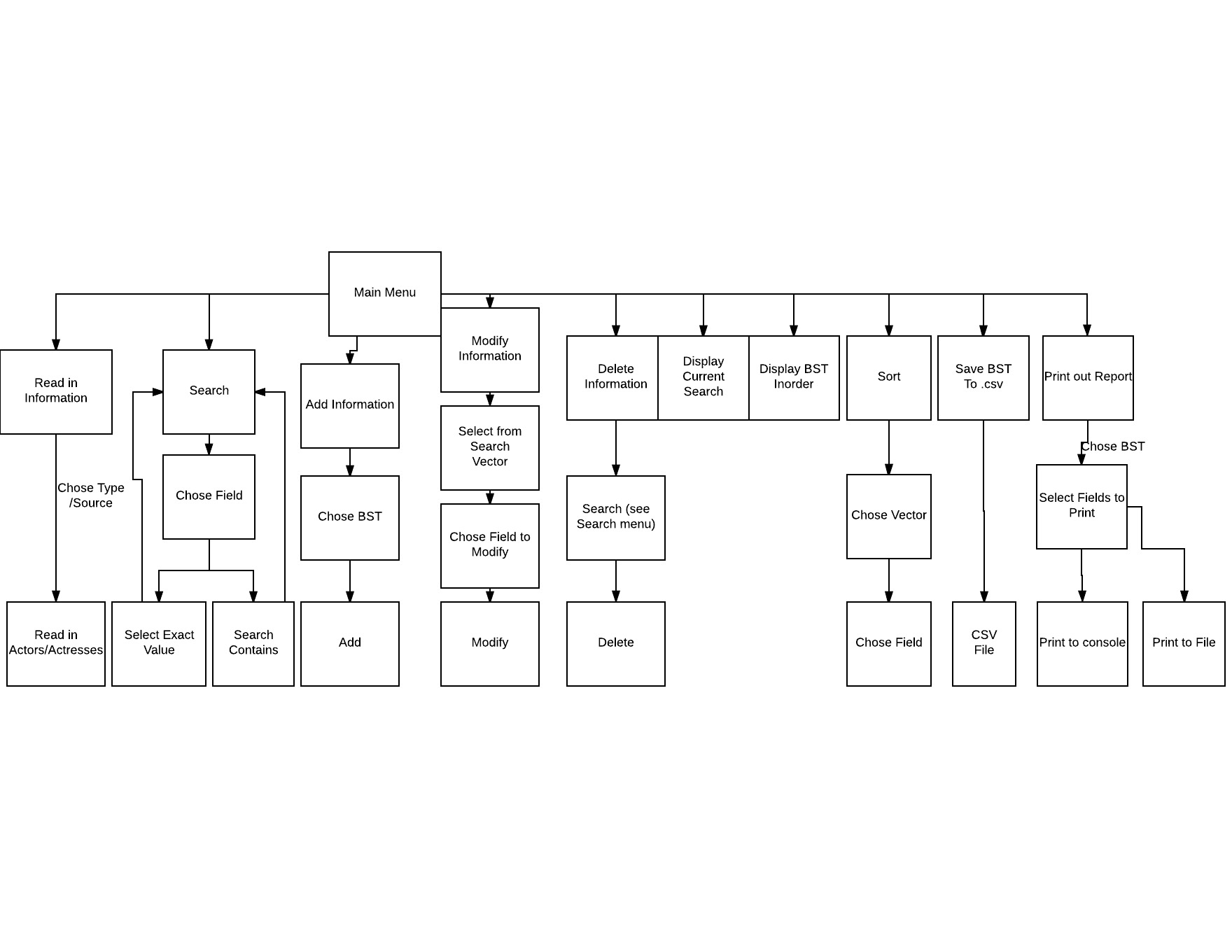
Also, the program has the ability to read .csv files with the appropriate number of fields (5 and 10) for actors/actresses and pictures, respectively.

5. Output: Give the user multiple options, including printing a report of either BST inorder to either a file or to the screen. The report can alternatively return the most current search vector. In both cases, the user will be able to choose which fields they would like to print.

6. The only function called by main() is the display menu function, which consists of a switch statement full of function calls. The first option is the going to be the option to read in a file, either from a default, or user specified. We will also allow the user to add individual nodes to either tree. The modify and delete functions will be dependent on the search function, as we will require the user to select the record to modify from the search list. This allows more precision for accessing individual members. (We'll be using this same file throughout the program, thus the user will be able to try to modify/delete entries, which we'll need to error check for). We will read in from a file, directly into the BST. The records are ordered by year, not by name, so we don’t have to worry about our tree becoming a linked list. Our search for exact name will therefore be O(log(n)). When the user indicates they want to search using a different criteria (or searching the name for contains), we will traverse the entire tree O(n) and add appropriate nodes to a vector O(n). If they want to narrow the search, we provide that option to search from the vector containing search results. We will enable the user to select sorted fields to print, either to the console, or to a file.

7. For reading the files into a BST, I decided (since our data is sorted by a different field in the files provided) that we do not need the extra complexity of reading to a vector, sorting, then adding recursively to the BST. For the search results, I considered arrays, vectors, linked lists, queues, primary queues, stacks, and heaps. Arrays are not dynamic by default, thus it would cause issues because we're not sure how big the return may be. Queues and Stacks, while they're easy to add to and remove from, are impossible to sort. Heaps and primary queues are automatically sorted, but lack any kind of direct access for modification or deletion of files. I am left with vectors and linked lists, which are both easy to add to (I would only be adding to the end of the vector, and thus would be dealing with a constant O() value). I can use the sort algorithm to sort a vector efficiently O(n(log(n)). The direct access of the vector allows me to grab any member designated by the user, for precise modification or deletion. Thus I will be using the BST for my main data structure, with vectors to store my search results

8.Menu interaction



9. Developed using Microsoft Visual Studio 2017  
Compiled on the CSE Grid.  
All requirements met.

Unfortunately I was unable to complete extra credit due to time constraints, as well as a problem with the .csv nominations file, where the values of the fields were not consistent, depending on type of award