***Bartholomew: The Academy Awards Database Robot***

Iris Vucic and Josh Mansito

Due: December 10th 11:59pm

**2: Problem Description**

This program is designed to give a user access to a database of records from the Academy Awards. Using a menu-based system, the user will be able to search through the entirety of the records. They will also be able to sort, modify, add, and delete records within the database as much as they want.

**3: Overall Software Architecture**

* main() : Main driver for the program that calls and runs all member functions from the classes.
* createList(): Reads in the actor-actress.csv file and stores the information in a tree. It will also sort the tree by name.
* picture\_Tree(): Reads in the pictures.csv file and stores the information in a tree. It will also sort the tree by the picture’s name.
* actorSearch() / picSearch(): Searches through the database of a chosen field.
* printDatabase():After all manipulations of the database are finished, this function will print the chosen database out to a .csv file.
* actorAdd() / picAdd(): Allows the user to add a record to the database chosen.
* actorModify() / picModify(): Allows the user to modify a record of the user’s choice. [not implemented]
* actorDelete() / picDelete(): Allows the user to delete a record from a chosen database.
* actorSort() / picSort(): Will sort the database by field, chosen by the user.
* errorCheck(): Will check user input for potential errors according to how many fields of input are required and whether or not these requirements are satisfied (not leaving out/overfilling input)

This project will be designed around a **menu** system that incorporates a lot of small menu functions and controls the flow of the program and user-controlled system.

We will house all the menu functions in two classes within Menu.cpp - actMenu for the Actor/Actress database and picMenu for the Pictures database. Both classes share the same menu functions, just with different output.

Current menu function implementation is as followed:

* main\_Menu(): Outputs a list of options that the user can perform on the database (e.g. add, delete)
* field\_Menu(): Case system for outputting variable fields the user can perform actions on according to the database (e.g. year/award/winner for Actors/Actress, title/duration/genre for Pictures)
* type\_Menu(): Case system for searching or editing a field
* confirmation\_Menu(): Case system to confirm the user’s input or choice.
* another\_Menu(): Case system for whether or not the user would like to perform an action again
* error\_Check(): Error checks user input to ensure they are satisfying parameter input requirements
* set\_Info(): Sets user input info to the variables of the database.
* award\_Selection(): Lets the user select an award category for the Actor/Actress tree
* choose\_Mode(): Allows the user to choose between a contains or partial search

**4. Input Requirements**

* **Input 1: Menu system**
  + Cin from keyboard, int, switch, valid input: 1 - 9. One character for valid input. There will be different valid inputs depending on the menu system being used by the user (which depends on the database).
* **Input 2: Inputs for the first tree (actors and actresses’ database) – all subject to change**
  + **actorAddRecord()**
    - * **These will all be cin from the user**
    - Add First Name:String | Will accept a “-“ style name up to 40 characters long.
    - Add Last Name:String | Will accept a “-“ style name up to 40 characters long
    - Add Win:Int | 1-character input | Formatted 1 for yes, they won an award, and 2 for no, they did not win one.
    - Add Film:String | Will accept special characters | Up to 70 characters long | adds the name of the film
    - Add Year: Int | 4 integer input | “0000” format.
    - Add Award**:** Int | 1-character input | This will be formatted to let the user select an award tied to a corresponding number. | Adds awards for actor/actress lead and supporting role.
    - Confirm Changes Made:Int | 1-character input | Confirm changes, edit changes, remove changes, represented by 1,2 and 3 respectfully.
    - More Changes / Exit:Int | 1-character input | Will let the user make more changes or exit the database with 1 for yes, 2 for no.
  + **actorDeleteRecord()**
    - * **These will all be cin from the user**
    - Choice of field: Int | 1-character input | Lets the user select which field they would like to modify
      * + Film – Name – Award – Winner – Year
    - Input for deletion: This will be different for each field.
      * + Film: String | Will accept special characters | Up to 70 characters long
        + Name: String | Will accept a “-“ style name up to 40 characters long.
        + Award: Int | 1-character input | “1 – 4” for the 4 options
        + Winner: Int | 1-character input | “1 – 2” for the 2 options
        + Year: Int | 4 integer input | “0000” format.
    - Choice of record in field: Int | 1-4-character input | “1 – 9999” | Lets the user select the record they want deleted
    - Confirmation: Int | 1-character input | “1 – 2” for yes or no
    - More Changes / Exit:Int | 1-character input | Will let the user make more changes or exit the database with 1 for yes, 2 for no.
  + **actorAModify() [not implemented]**
    - * **These will all be cin from the user**
    - Choice of field: Int | 1-character input | Lets the user select which field they would like to modify
      * + Film – Name – Award – Winner – Year
    - Input for Modification: This will be different for each field.
      * + Film: String | Will accept special characters | Up to 70 characters long
        + Name: String | Will accept a “-“ style name up to 40 characters long.
        + Award: Int | 1-character input | “1 – 4” for the 4 options
        + Winner: Int | 1-character input | “1 – 2” for the 2 options
        + Year: Int | 4 integer input | “0000” format.
    - Choice of record in field: Int | 1-4-character input | “1 – 9999” | Lets the user select the record they want to modify
    - Choice of modification: Int | 1-character input | Lets the user select which field they would like to modify
      * + Film – Name – Award – Winner – Year
    - Input for Modification one in record: This will be different for each field.
      * + Film: String | Will accept special characters | Up to 70 characters long
        + Name: String | Will accept a “-“ style name up to 40 characters long.
        + Award: Int | 1-character input | “1 – 4” for the 4 options
        + Winner: Int | 1-character input | “1 – 2” for the 2 options
        + Year: Int | 4 integer input | “0000” format.
    - Confirmation: Int | 1-character input | “1 – 2” for yes or no
    - More Changes / Exit:Int | 1-character input | Will let the user make more changes or exit the database with 1 for yes, 2 for no
  + **actorSearch() – searching through the database**
    - * **These will all be cin from the user, and per the instructions, there will a complete “exact” search, and a “contains” partial search.**
  + **Complete search using “exact” method:**
    - Choice of field: Int | 1-character input | Lets the user select which field they would like to modify
      * + Film – Name – Award – Winner – Year
    - Input for Searching: This will be different for each field.
      * + Film: String | Will accept special characters | Up to 70 characters long
        + Name: String | Will accept a “-“ style name up to 40 characters long.
        + Award: Int | 1-character input | “1 – 4” for the 4 options
        + Winner: Int | 1-character input | “1 – 2” for the 2 options
        + Year: Int | 4 integer input | “0000” format.
    - Choice of record in field with “exact” match method: Int | 1-4-character input | “1 – 9999” | Lets the user select the record they want to search
    - Confirmation: Int | 1-character input | “1 – 3” for yes, no, search again
  + **Partial search using “contains” method:**
    - Choice of field: Int | 1-character input | Lets the user select which field they would like to modify
      * + Film – Name – Award – Winner – Year
    - Input for Searching: This will be different for each field.
      * + Film: String | Will accept special characters | Up to 70 characters long
        + Name: String | Will accept a “-“ style name up to 40 characters long.
        + Award: Int | 1-character input | “1 – 4” for the 4 options
        + Winner: Int | 1-character input | “1 – 2” for the 2 options
        + Year: Int | 4 integer input | “0000” format.
    - Choice of record in field with “contains” match method: Int | 1-4-character input | “1 – 9999” | Lets the user select the record they want to search
    - Confirmation: Int | 1-character input | “1 – 3” for yes, no, search again
  + **actorSort() – sorting the database**
    - * **These will all be cin from the user**
    - Choice of Sort: Int | 1-character input | Lets the user select which field they would like to sort by
      * + Film – Name – Award – Winner – Year
    - Confirmation: Int | 1-character input | “1 – 3” for yes, no, sort again
  + **printDatabase() – printing the database**
    - * **This will be cin from the user**
    - Print Database: Int | 1-character input | “1 – 2” for yes or no
* **Input 3: Inputs for the second tree (Pictures database) – all subject to change**
  + **pictureAddRecord()**
    - * **These will all be cin from the user**
    - Add Nomination: Int | 1-2-Character input | Lets user add nominations up to 99 to a film
    - Add Rating:Double | 3-4-Character input | Lets user add IMDB rating to movies with 0.0-10.0 scale system
    - Add Meta Rating:Int | 1-3-Character input | Lets user add Meta rating to movies with 0 – 100 scale system
    - Add Duration:Int | 2-3-Character input | Lets user add time duration in minutes up to 999 for the film
    - Add Genre Menu:Int | 1-Character input | 1-Character input | This will allow the user to select a number that corresponds with a genre to choose from 1 – 8(or 9 if we have “other”)
    - Add Synopsis:String | User input until \n is pressed | Add a synopsis to the movie
    - “Other” selection from Genre Menu:String | Lets the user type in the genre they want up to 30 characters
    - User typed Genre Select:String | Lets the user type in the genre they want up to 30 characters
    - Add Release:Int | 1-2-Character input | 1 – 12 for each month of the year added
    - Add Film:String | Will accept special characters | Up to 70 characters long | adds the name of the film
    - Add Year: Int | 4 integer input | “0000” format.
    - Confirm Changes Made:Int | 1-character input | Confirm changes, edit changes, remove changes, represented by 1,2 and 3 respectfully.
    - More Changes / Exit:Int | 1-character input | Will let the user make more changes or exit the database with 1 for yes, 2 for no.
  + **picDeleteRecord()**
    - * **These will all be cin from the user**
    - Choice of field: Int | 1-character input | Lets the user select which field they would like to modify
      * + Nomination – IMDB rating – META rating – Duration – Genre – Synopsis – Release – Film – Year *(Note: The order of these may be changed, I just wanted to match the way I ordered them in the add\_Record() section)*
    - Input for deletion: This will be different for each field.
      * + Nomination: Int | 1-2-Character input | “0 – 99”
        + IMDB: Double | 3-4-Character input | “0.0-10.0”
        + META: Int | 1-3-Character input | “0 – 100”
        + Duration: Int | 2-3-Character input | “0 – 999”
        + Genre: String | 1-30-Character input | Not entirely sure how to do this yet with special characters
        + Synopsis: String | \n terminated input | Not entirely sure how to do this yet with special characters
        + Release: Int | 1-2-Character input | “1-12”
        + Film: String | Will accept special characters | Up to 70 characters long
        + Year: Int | 4 integer input | “0000” format.
    - Choice of record in field: Int | 1-4-character input | “1 – 9999” | Lets the user select the record they want deleted
    - Confirmation: Int | 1-character input | “1 – 2” for yes or no
    - More Changes / Exit:Int | 1-character input | Will let the user make more changes or exit the database with 1 for yes, 2 for no.
  + **picModifyRecord() [not implemented]**
    - * **These will all be cin from the user (This process will need to be done twice. First for selecting the record, then again for selecting the field to modify)**
    - Choice of field: Int | 1-character input | Lets the user select which field they would like to modify
      * + Nomination – IMDB rating – META rating – Duration – Genre – Synopsis – Release – Film – Year *(Note: The order of these may be changed, I just wanted to match the way I ordered them in the add\_Record() section)*
    - Input for Modification: This will be different for each field.
      * + Nomination: Int | 1-2-Character input | “0 – 99”
        + IMDB: Double | 3-4-Character input | “0.0-10.0”
        + META: Int | 1-3-Character input | “0 – 100”
        + Duration: Int | 2-3-Character input | “0 – 999”
        + Genre: String | 1-30-Character input | Not entirely sure how to do this yet with special characters
        + Synopsis: String | \n terminated input | Not entirely sure how to do this yet with special characters
        + Release: Int | 1-2-Character input | “1-12”
        + Film: String | Will accept special characters | Up to 70 characters long
        + Year: Int | 4 integer input | “0000” format.
    - Choice of record in field: Int | 1-4-character input | “1 – 9999” | Lets the user select the record they want modify / then during second time the field they want to modify
    - Confirmation: Int | 1-character input | “1 – 2” for yes or no
    - More Changes / Exit:Int | 1-character input | Will let the user make more changes or exit the database with 1 for yes, 2 for no.
  + **picSort() – sorting the database**
    - * **These will all be cin from the user**
    - Choice of field: Int | 1-character input | Lets the user select which field they would like to sort by
      * + Nomination – IMDB rating – META rating – Duration – Genre – Synopsis – Release – Film – Year *(Note: The order of these may be changed, I just wanted to match the way I ordered them in the add\_Record() section)*
    - Confirmation: Int | 1-character input | “1 – 3” for yes, no, sort again
  + **picSearch() – searching through the database**
    - * **These will all be cin from the user, and per the instructions, there will a complete “exact” search, and a “contains” partial search.**
  + **Complete search using “exact” method:**
    - Choice of field: Int | 1-character input | Lets the user select which field they would like to search
      * + Nomination – IMDB rating – META rating – Duration – Genre – Synopsis – Release – Film – Year *(Note: The order of these may be changed, I just wanted to match the way I ordered them in the add\_Record() section)*
    - Input for search: This will be different for each field.
      * + Nomination: Int | 1-2-Character input | “0 – 99”
        + IMDB: Double | 3-4-Character input | “0.0-10.0”
        + META: Int | 1-3-Character input | “0 – 100”
        + Duration: Int | 2-3-Character input | “0 – 999”
        + Genre: String | 1-30-Character input | Not entirely sure how to do this yet with special characters
        + Synopsis: String | \n terminated input | Not entirely sure how to do this yet with special characters
        + Release: Int | 1-2-Character input | “1-12”
        + Film: String | Will accept special characters | Up to 70 characters long
        + Year: Int | 4 integer input | “0000” format.
    - Choice of record in field with “exact” match method: Int | 1-4-character input | “1 – 9999” | Lets the user select the record they want to search
    - Confirmation: Int | 1-character input | “1 – 3” for yes, no, search again
  + **Partial search using “contains” method:**
    - Choice of field: Int | 1-character input | Lets the user select which field they would like to search
      * + Nomination – IMDB rating – META rating – Duration – Genre – Synopsis – Release – Film – Year *(Note: The order of these may be changed, I just wanted to match the way I ordered them in the add\_Record() section)*
    - Input for search: This will be different for each field.
      * + Nomination: Int | 1-2-Character input | “0 – 99”
        + IMDB: Double | 3-4-Character input | “0.0-10.0”
        + META: Int | 1-3-Character input | “0 – 100”
        + Duration: Int | 2-3-Character input | “0 – 999”
        + Genre: String | 1-30-Character input | Not entirely sure how to do this yet with special characters
        + Synopsis: String | \n terminated input | Not entirely sure how to do this yet with special characters
        + Release: Int | 1-2-Character input | “1-12”
        + Film: String | Will accept special characters | Up to 70 characters long
        + Year: Int | 4 integer input | “0000” format.
    - Choice of record in field with “contains” match method: Int | 1-4-character input | “1 – 9999” | Lets the user select the record they want to search
    - Confirmation: Int | 1-character input | “1 – 3” for yes, no, search again
  + **printDatabase() – printing the database**
    - * **This will be cin from the user**
    - Print Database: Int | 1-character input | “1 – 2” for yes or no
* **5. Output Requirements**
* **Outputs for BOTH trees**
  + printDatabase():After all manipulations of the database are finished, this function will print the chosen database out to a actDatabaseFinal.csv file.
    - The following will update the database depending on user decisions then update the database file
      * addRecord():
      * modifyRecord():
      * deleteRecord():
    - The following will update the database depending on the sort method used, then update the database file
      * sortDatabase():
* **Menu Outputs:** All menu calls will output different text to the screen depending on the database the user is in. These texts will include listing fields the user can add to/delete/sort, as well as confirmation menus to ensure the user has not made a mistake when adding input.
* **All output to console:** This was all done with a combination of string, ints, and function calls. The menu system handles the text based navigation for the user, and the functions will send out the data output as needed.

**6: Problem Solution Discussion**

**Sorting the database**

* Upon initially reading in the .csv files, the preexisting entries in the files will be automatically read into an array and sorted using quicksort. After sorting the entries, they will be loaded into binary search trees. For every additional user entry, modification, or deletion in the database, the trees will be automatically re-sorted after the operation using quicksort as well.

**Complete search using “exact” method**

* Since we will be implementing the database using binary search trees, to traverse and search the tree for a value that matches the exact search entry, we will use a quicksort algorithm. Not only is quicksort the fastest algorithm, but it is also recursive and will search faster with binary search trees because the entries in the trees have already been sorted when the trees were created or last modified.

**Partial search using “contains” method**

* Since we are going to be implementing the database using binary search trees, to search a substring of an entry (a value that “contains” the entry), the entries of the trees will be divided into arrays. The search entry will also be converted to an array. Within these arrays, the characters of the tree arrays will be searched for a match of characters in the search entry.

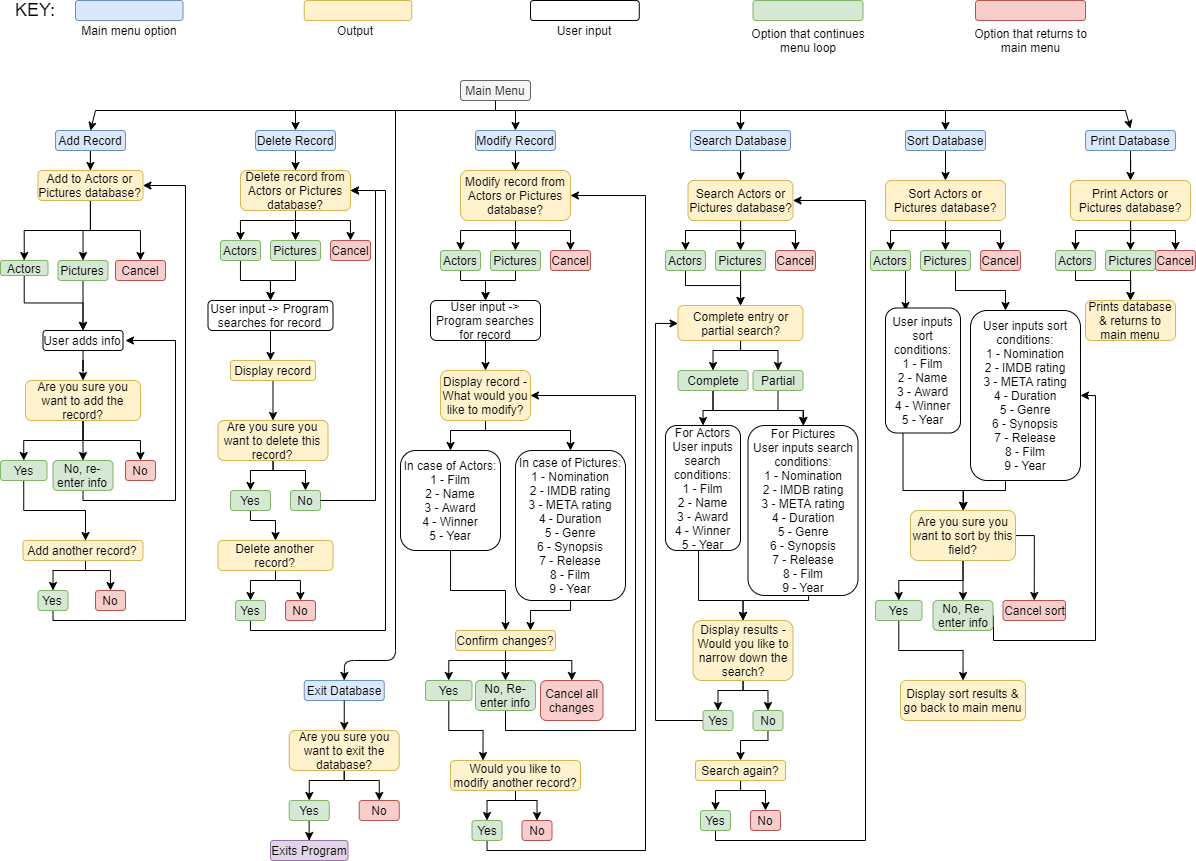
**Adding additional message for Delete function**

* This function required a complete rework of a lot of functions. Our solution ended up adding special search functions that are JUST for the delete function. We were also required to add a node\_Remover function to our BSTree.h. This function did the main tree manipulation for the deletion. The method we used for traversing the tree for deletion was check the node -> Left -> Right. The source for this logic is: [Deletion from BST (Binary Search Tree) - Techie Delight](https://www.techiedelight.com/deletion-from-bst/).

**7: Data Structures**

We are planning to implement our database using binary search trees and arrays. Additionally, we intend to implement the quicksort algorithm for both sorting the trees and searching through them. The advantages to implementing our database using binary search trees are that they provide an optimal solution for the most important component of the program: searching the database. Binary search trees are already sorted, so when the user is searching for an entry whether it be a whole string or a substring, the database will be able to search the trees and return an answer quickly. We also chose binary search trees over another potential solution, AVL trees, because binary trees are more malleable. AVL trees always require a balance condition, which could conflict with the database because the .csv input or user entries provided may not always be able to create a balanced tree if there are empty fields. Through binary search trees, the user is easily able to add, modify, or delete an entry and sort it into the tree without having to keep track of balance. The only downside to using binary search trees is the necessity to copy the database just to reorder a single tree. Using arrays is also beneficial to the program for the purpose of searching using the “contains” method in order to ensure a substring of a leaf in the tree is searchable. Other than this main benefit, arrays will not really be used as they are not as effective or fast as binary search trees.

**8: User Interface Scheme**



**9: Status of Application**

The project was developed on Visual Studio and Clion. It compiled and operated successfully on csegrid. The status of the project:

* Successfully compiling
* Modify is not implemented for either database
* We are having a few issues with some of the search mechanics, but overall it is functional
  + Bug sometimes happening with film field
* Delete function is operational but when doing a partial search with the big file, it needs the search to be narrowed down. It was written while using our smaller test file (which automatically narrowed the search since it limited the amount of entries to look through)
* Our main logic that we started with when we originally wrote the design document seemed to work out for us for the most part. There has of course been a lot of modifications, name changes, additions, and removed content throughout the program creation, however, the majority of the logic remained the same.