Gregory Wicklund

Student ID: 810906774

Class: CSCI 2421

Final Report

For my final project I am reading the information regarding Academy Award winners in to two binary search trees (BSTs) using two given files: pictures.csv and actor-actress.csv. With this information we need to build two binary search trees, and include functions for adding, removing, searching, sorting, and printing the information from these BSTs.

I ended up with quite a few more files than I had anticipated, due to how I organized my program. I used a philosophy that my old Linux teacher had said: “If you need to do something more than once, write a script for it!” I also separated out the functions that might be working with Actor or Picture objects, but didn’t necessarily need access to all the member variables of trees or nodes.

Status of program:

All non-menu functions are working properly and can handle user error. Menu functions will send the program into an infinite loop if a non-numeral is entered.

Both Actor and Picture binary search trees can be successfully created, sorted, modified, and printed.

Program compiles and runs successfully on the csegrid.

(Images for software architecture and menu flow are included as SoftwareArchitecture.png and MenuFlow.png, respectively).

Design Document

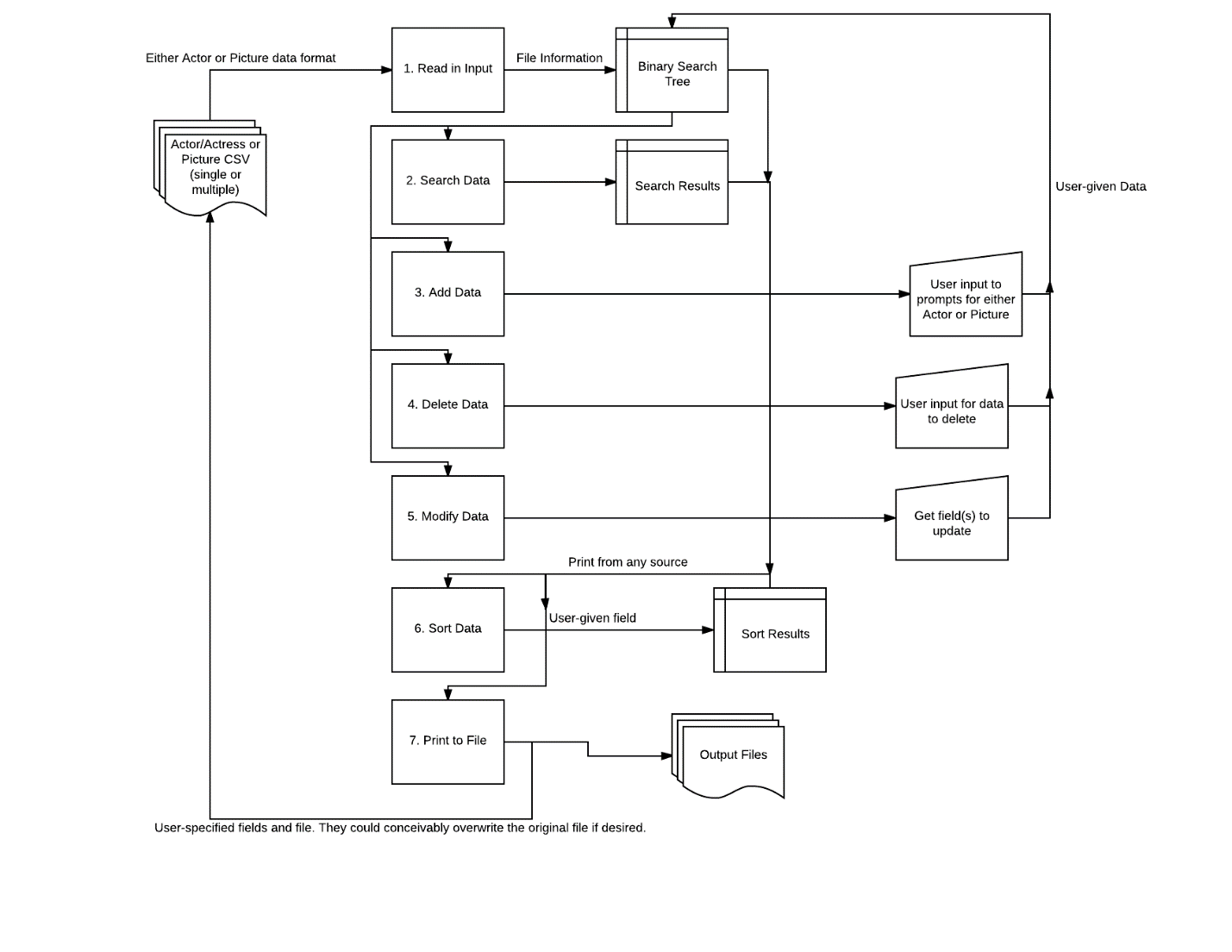
1. Oscars Database

Gregory Wicklund

Student ID: 810906774

1 December 2017

1. Reads in information relating to Academy Award nominees in one of two formats, creates a binary search tree of the information read, and allows the user to search the tree, sort the data, add/delete/modify records, and print either the full database, search results, or the sorted database/search results.
2. Overall software architecture



1. Input requirements:

Two main data types: Actor and Picture

Actor

1. Name – string, no character limit, any characters accepted
2. Year – int, no character limit, only accepts numerals (0-9)
3. Award – string, no character limit, any characters accepted
4. Film – string, no character limit, any characters accepted
5. Winner – bool, user selects choice from menu, file reads either. No direct user input

Picture

1. Name – string, no character limit, any characters accepted
2. Year – int, no character limit, only accepts numerals (0-9)
3. Nominations – int, no character limit, only accepts numerals (0-9)
4. Rating – double, no character limit, only accepts numerals (0-9) and decimal (.)
5. Duration – int, no character limit, only accepts numerals (0-9)
6. Genre 1 – int, no character limit, any characters accepted
7. Genre 2 – int, no character limit, any characters accepted
8. Release – string, user selects month of choice from menu, file contains string. No direct user input
9. Metacritic – int, no character limit, only accepts numerals (0-9)
10. Synopsis – string, no character limit, any characters accepted

User can search any field of either datatype, search within searches, sort, and add/delete/modify records.

For reading in, the program assumes the first line of the input file will contain the “key” and check it to make sure the file is of correct type. This prevents unpredictable behavior if a different file is read in. (usually crashing but better to be safe)

1. Output types from user:

User can output either the full binary search tree into a csv; or search/sort results into a table, with any fields able to be toggled on and off. These fields are the same types as the inputs, which the exception of the Winner Boolean, which outputs a 0 or 1 with the full tree output, and “Yes” or “No” with the results output.

Full tree output:

Fields are separated by a ‘,’ and records by a newline

Actor results output:

Fields are separated by a ‘|’ and records by a newline

Picture results output:

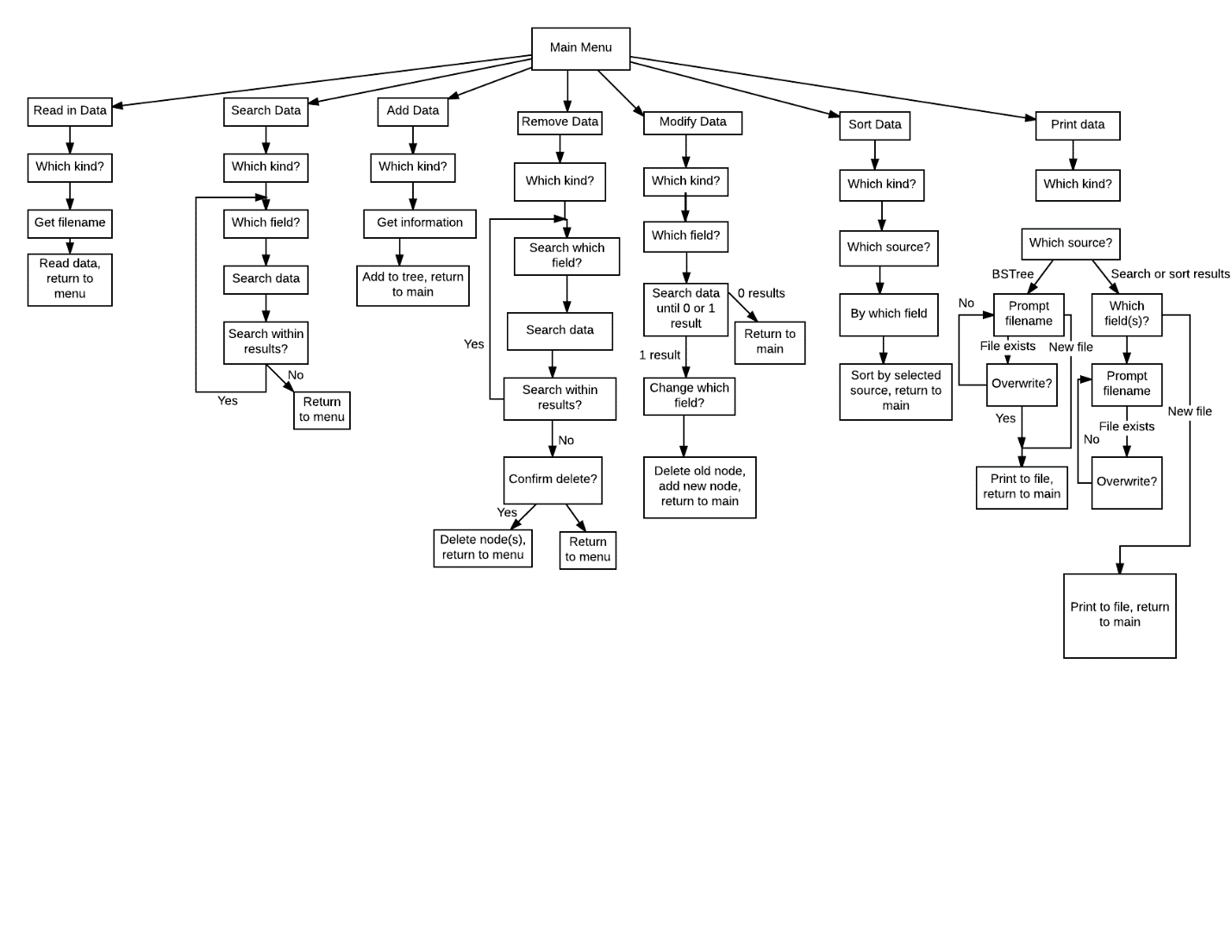
Records are separated by a series of ‘--------’ and fields are labeled. I chose this because the length of the synopses would make a table format quite difficult to read.

1. Problem Solution Discussion

Reading in the information was fairly straightforward: I used a series of getline commands to parse the data, ensure it was in the proper format, and add the created record to the tree. For my searches and sorts, I stored my results to a vector of appropriate type (Actor or Picture) and returned that vector, for further searching/sorting or for printing. Non-key searches and all sorts used an inorder traversal to visit every node on the tree, and non-key sorts used the STL sort with a custom comparison function to compare the appropriate fields.

I had to be a bit creative with reading in from the files, since there were situations where a comma existed in the middle of a field, usually surrounded by quotation marks. Further though, there were some situations where there were quotation marks in the middle of the field that were part of the field, which necessitated a str.replace() function to replace double-quotes (“”) with single quotes (“).

1. I elected to use vectors to hold my search results and sort results. I considered a vector and priority queue for sort results, but realized a priority queue would not be a good choice, considering how often I might need to access data in the middle of the set. I also briefly considered linked lists, but elected against it, again because of needing to access data in the middle of the set. These also let me pass specific elements in the vector to other functions as needed.
2. User Interface Scheme



1. Status of applications

IDE Used: Visual Studio 2017

Compiled successfully using Ubuntu on Windows Subsystem for Linux and csegrid

So far as I know, all requirements on the final project have been met.

Known issues:

-Delete and Modify should only modify a node if the first one found matches the record to be deleted/modified. A way around this would be to write a function that can continue searching from that found node if it was not the correct one.

-Menus can only travel forward, there is no way to step back from a menu without either getting through all the prompts or restarting the program.

-Some menus may be a bit vague in what they are asking.

-Sorts can only be done in ascending order. I have a commented-out function template in NodeActor.cpp and ActorFunctions.cpp that can be adapted for whichever datatype is needed for a descending-order search. Alternatively, using the reverse() function on an already-sorted vector would accomplish the same, but would be much more expensive computation-wise. (adding another n/2 cycles, where n is half the size of the vector)

-There seems to be a very occasional crash if trying to read in from the corrected nominations.csv file.

Summary of functions:

Main Menu give options for reading from file, searching the database, adding/removing objects from the database, and printing either the full database with any changes (sorted by name) or the sort/search results to a file. The full database retains the csv formatting: sort/search results are in table form for Actors/Actresses, and grouped into separate entries for Pictures (for clarity, because of how long the synopses can get).

Read Data In

Prompts to read either from Actors/Actresses or Pictures, then prompt for an input file. It then checks to make sure the first line matches the expected form, and if so reads in the data from that input file to a binary search tree of appropriate type. If a tree exists, the program will append the data to that tree, allowing for multiple input files to be used. I didn’t get one running in time for the project, but I’d like to have a duplicate guard running to prevent multiple leaves with the exact same data, or if the user accidentally reads in the same file twice. Originally I was going to read in both at the start of the program, however I elected to make it user-controlled. This saves on memory and computation time, if the user didn’t want to work with both datasets, and also lets them control exactly which files get read in, instead of only using the one.

Search data for specific keyword

Prompts to search from either the Actor/Actress or Picture binary search tree. Checks to make sure there’s data in the tree (no need to run a search on an empty tree), then prompts for which field to search. For all except picture release month, prompts for a search string, then for complete or partial. For picture release month, asks which month to search. Searches the binary search tree by the user-specified field.

Complete searches using the tree’s key (actor/actress name or picture name) use the given search node function, and traverse the tree to find all matches. Partial key searches, along with complete and partial of any other field, traverse the entire tree, using an inorder traversal, to visit every node and search its data for the search term. I store the search results in a vector, since it’s the most versatile data form for what I need (search again, sort, or print).

After the first search is done, the program will prompt if the user wishes to do a second search, again getting the search term. This time, though, it will search the vector, searching each element in turn. I have it set up to continue until the user is either satisfied or there are no more results.

Add data to file

Prompts for either a new Actor/Actress or Picture, and then prompts for each field in turn. Adds a new node to the tree with the user-supplied data. The program is set up in such a way that the node can be added to a tree of any size (provided there is enough memory to allocate).

Delete data from file

Runs a search to get node(s) to delete, then runs in a for loop to delete these nodes. At the moment a node can only be deleted if it’s the first one found, otherwise the program will skip deleting it. I check this by making sure the found node’s data matches the data of the node the user wants deleted. I think if I coded a second search function that could take a node argument, and use that as “root” for a new search I could find the appropriate node, if it exists, using a similar method with the delete function to delete it.

Sort Data

Program will prompt for which type of object to sort (Actor/Actress or Picture) and which data set they wish to use (the full tree, search results, or existing sort results). It will then ask which field the user wishes to sort by, and then sort by that field in ascending order. A vector with the sorted data is returned, separate from the vector for search results.

Save current data to file

Program will prompt which set of data to save: the current tree (with additions/removals), search results vector, or the sorted vector. For the current tree, it will save it back in the same csv format as it was read in as. For the search results and sorted vectors, it will prompt which fields to use, and then generate a table with the appropriate fields, printing it to a user-given file.

Exit

Leaves the main menu, which will cause the tree and vectors to fall out of scope, and exits the program from main.