Program 5

Andreas Neacsu

April 25, 2023

1. Program Description

1.1. Program Requiremnts

Write a program to hold information for a small TV show database using a binary search tree. Specifically, the show names, broadcast years, category, and the actor/actress names.

1.1.1. Additional Requirements

→ Display all shows in the tree (only the titles!).

→ Display all actors of a given show in the tree: NCIS, McHale's Navy, The Prisoner, The Office and two others of your choice.

→ Display all shows of a given actor: Bill Daily, Dana Elcar, Andy Griffith, Tress MacNeille and two others of your choice.

→ Display all shows released between 2005 and 2015 and one other decade range of your choice.

1.2. Input

The input for this program is a .txt file formated as follows

```
Series Title (Series Begin - Series end)
Series Genre
IMDB Link
Actors
```

2. Classes and Structures

2.1. Classes

2.1.1. Binary Search Tree

The BST class includes various functions that can be used to perform different operations on the tree data structure, including inserting nodes, searching for nodes, and displaying node data. The implementation uses a binary search tree structure, where nodes are inserted based on he value of a specific field (in this case, the "series_name" field of the "movie_data" struct).

2.2. Structures

2.2.1. movie_data

The movie_data structure is designed to hold all information necessarry for one entry. It holds the series title as a std::string, 2 integers that represent the start and end year, a std::string that hold the series category, a std::string that holds the series URL, a std::vector<std::string> that holds all the actors of the series and an integer that represents the nodes index.

2.2.2. tree_node

The tree_node structure defines the basis of out BST. It holds 2 tree_node pointers that represent its left and right child respectivley and a movie_data to store all the information in one place.

3. FUNCTIONS

The functions used for program 5 are described below

3.1. file_reader.cpp

3.1.1. std::vector<movie_data> create_movie_data_vector()

This function reads a file containing TV series data and returns a vector of movie_data objects

3.1.2. void get_series_name(char[], char[])

This function extracts the series name from a line of text

3.1.3. void get_years(char[], int&, int&)

This function extracts the sereis years from a line of text

3.1.4. int index_of(char[], char[])

This function returns the index of the first occurrence of character 'x' in string 'y'.

3.1.5. void get_subString(char[], int, int, char[])

This function extracts a substring from 's' and stores it in 'res', starting at index 'start' and ending at index 'start+end-1'.

3.1.6. void remove_blanks(char[])

This helper function removes any trailing whitespace (spaces or tabs) from a string.

3.2. tree.cpp

3.2.1. void insert_node_private(tree_node*&, tree_node*)

Private member function that recursively inserts a node in the tree

3.2.2. void display_data_private(tree_node*)

Private member function that recursively displays the data in the tree

3.2.3. tree_node* search_series_name_private(std::string, tree_node*)

Private member function that recursively searches for a node with a given series name

3.2.4. void get_series_vector_from_actor(std::vector<std::string>&, tree_node*, std::string)

This function searches for all the TV series present in the tree that a particaular actor has acted in and adds them to the shows_of_actor vector.

3.2.5. void get_series_vector_from_range(std::vector<std::string>&, tree_node*, int, int)

This function searches for all the TV series present in the tree that were aired between a particular range of years and adds them to the series_of_range vector

3.2.6. Tree()

Constructor for the Tree class. It initializes the root_node to NULL and creates a vector of movie_data objects. Then, it creates a tree based on the data in the vector

3.2.7. ~Tree()

Destructor for the Tree class. It deallocates the memory used by the root node.

3.2.8. void create_tree(std::vector<movie_data>)

This function creates a tree using the given vector of movie_data objects. It creates a new tree_node for each movie_data object and inserts it into the tree.

3.2.9. tree_node* create_node(movie_data)

This function creates a new tree_node object with the given movie_data object. it returns a pointer to the newly created node.

3.2.10. void insert_node_public(tree_node*)

This function inserts a node into the tree. It recursively searches the tree to find the correct position for the new node.

3.2.11. tree_node* search_series_name_public(std::string)

This function searches for a particular TV series name in the tree and returns the node that contains that TV series.

3.2.12. void get_actor_from_series(std::string)

This function searches for a TV series in the tree and displays all the actors who have acted in it

3.2.13. bool search_actor_list_of_node(tree_node*, std::string)

This function searches for a particular actor in the actor vector of a given node. It returns true if the actor is present in the vector, and false otherwise.

3.2.14. void get_series_from_actor(std::string)

This function prints out all the TV series present in the tree that a particular actor has acted in

3.2.15. void get_series_within_range(int, int)

This function searches for all the TV series present in the tree

3.2.16. void display_data_public()

This function displays the data present in the tree to the user.

3.2.17. void display(tree_node*)

This function displays the TV series name of the current node.

3.2.18. void display_actors(tree_node*)

This function displays the names of actors associated with a particular TV series stored in the node pointed to by displayable_node.

3.3. main.cpp

3.3.1. void display_database(Tree)

This function displays the entire database of TV series in the tree. It calls the public display_data_public() function of the Tree class to accomplish this.

3.3.2. void get_series_of_actor(Tree)

This function prompts the user to enter the name of an actor and then calls the get_series_from_actor function of the provided tree object to find all the TV series that the actor has acted in.

3.3.3. void get_actors(Tree)

This function gets the actors who have acted in a particular TV series.

3.3.4. void get_series_of_range(Tree)

This function prompts the user to input a range of years and then calls the get_series_within_range function of the Tree class to retrieve all the TV series that aired within that range.

3.3.5. void database_interface(Tree)

This function acts as an interface for the user to interact with the database stored in the tree.

4. ANNOTATED PROGRAM

4.1. tree.h

```
/* tree.h
 *
 * CS 121.Bolden.......Compiler Version: 12.0.5...........Andreas Neacsu
 * 4/26/2023, ............M1 Macintosh.......neac9115@vandals.uidaho.edu
 *
 * Header file for tree class, includes function / structure declarations
 *-------------------------------------------------------------------------
 */
/*
This code defines a header file named ‘‘tree.h’’ that contains the class
definition
for a binary tree data class. The header file includes necessary dependencies
and contains definitions for public and private functions used to initialize
the tree
database, search for and retrieve data from the tree, and display the data to
the user.
The binary tree consists of tree nodes with movie data and pointers to their
left and right child nodes.
*/
#ifndef TREE_H
#define TREE_H
#include <iostream>
#include ‘‘../file_reader/file_reader.h"

// Definition of the tree_node struct
struct tree_node{
 movie_data node_data;
 tree_node* left_child;
 tree_node* right_child;
};

class Tree
{
 private:
```

```
 tree_node* root_node;
// Definitions for the for the private functions used in initializing the tree
database
 void insert_node_private(tree_node*&, tree_node*);

// Definitions for private functions used for display and program requirement 0
 void display_data_private(tree_node*);

// Definitions for the private functions used for program requirement one
 tree_node* search_series_name_private(std::string, tree_node*);

// Definitions for the private functions used for program requirement two
 void get_series_vector_from_actor(std::vector<std::string>&, tree_node*,
std::string);

// Definitions for the private functions used for program requirement three
 void get_series_vector_from_range(std::vector<std::string>&, tree_node*, int,
int);

 public:
 ~Tree();

// Definitions for the for the public functions used in initializing the tree
database
 Tree();
 void create_tree(std::vector<movie_data>);
 void insert_node_public(tree_node*);
 tree_node* create_node(movie_data);

// Definitions for the public functions used for program requirement one
 tree_node* search_series_name_public(std::string);
 void get_actor_from_series(std::string);

// Definitions for the public functions used for program requirement two
 bool search_actor_list_of_node(tree_node*, std::string);
 void get_series_from_actor(std::string);

// Definitions for the public functions used for program requirement three
 void get_series_within_range(int, int);

// Definitions for the public functions used for display and program
requirement 0
 void display_data_public();
 void display(tree_node*);
 void display_actors(tree_node*);
};
#endif
```

4.2. tree.cpp

```
/* tree.cpp
 *
 * CS 121.Bolden.......Compiler Version: 12.0.5...........Andreas Neacsu
 * 4/26/2023, ............M1 Macintosh.......neac9115@vandals.uidaho.edu
 *
 * This code defines the implementation of the Tree class declared in the
 * "tree.h" header file.
 *-------------------------------------------------------------------------
 */
/*
This code includes various functions that can be used to perform different
operations on the tree data structure, including inserting nodes, searching for
nodes, and displaying node data. The implementation uses a binary search tree
structure, where nodes are inserted based on
the value of a specific field (in this case, the ‘‘series_name’’ field
of the ‘‘movie_data’’ struct). The insert_node_private() function is
used to recursively insert nodes into the tree based on the value of this
field. The display_data_private() function is used to recursively display
the data of each node in the tree.The Tree class also includes other
functions to search the tree for nodes based on different criteria, such as
search_series_name_private(), which searches for a node based
on the series name, and get_actor_from_series(), which displays the actors for
a given series.The code also includes various debugging statements that can
be enabled or disabled by commenting out the #define DEBUG statement at the
beginning of the file. Overall, this code provides a useful implementation of a
binary search tree data structure for storing and searching movie data.
*/

// The ifndef directive checks if the TREE_CPP header has already been included
in the file
// If not, it includes the header and defines TREE_CPP to avoid multiple
inclusion
#ifndef TREE_CPP
#define TREE_CPP

// DEBUG is a preprocessor macro used to enable/disable debug statements
// Here, it is commented out to disable debug statements
//#define DEBUG

// Include the header file for the Tree class
#include "tree.h"

// Include the necessary standard libraries
#include <iostream>

// Private member function that recursively inserts a node in the tree
void Tree::insert_node_private(tree_node*& target, tree_node* insertable_node)
{

// DEBUG statement that prints a message when the function is entered
 #ifdef DEBUG
 std::cout << ‘‘[DEBUG][TREE::insert_node_private] Entering’’ << std::endl;
 #endif
```

```
// If the target node is null, set it to the insertable node
 if(target == NULL ){
 target = insertable_node;
 return;
 }

// If the series name of the insertable node is the same as the target node, do
nothing
 if (insertable_node->node_data.series_name == target->node_data.series_name)
 return;

// If the series name of the insertable node is less than the target
node's series name, insert it in the left subtree
 else if( insertable_node->node_data.series_name <
target->node_data.series_name )
 insert_node_private(target->left_child, insertable_node);

// Otherwise, insert it in the right subtree
 else
 insert_node_private(target->right_child, insertable_node);

// DEBUG statement that prints a message when the function is exited
 #ifdef DEBUG
 std::cout << ``[DEBUG][TREE::insert_node_private] Exiting'' << std::endl;
 #endif
}

// Private member function that recursively displays the data in the tree
void Tree::display_data_private(tree_node* entry)
{

// If the entry is not null, recursively display the left subtree, the node,
and the right subtree
 if(entry != NULL)
 {
 display_data_private(entry->left_child);
 display(entry);
 display_data_private(entry->right_child);
 }
}

// Private member function that recursively searches for a node with a given
series name
tree_node* Tree::search_series_name_private(std::string key, tree_node* n)
{

// If the node is not null, check if its series name matches the given key
 if(n != NULL){
 if(n->node_data.series_name == key)
 {

// DEBUG statement that prints a message when the function finds the entry
 #ifdef DEBUG
 std::cout << ``[DEBUG][TREE::search_series_name_priv] Found Entry '' <<
std::endl;
```

```
 #endif
 return n;
 }

// If the key is less than the node's series name, search the left subtree
 else if(key < n -> node_data.series_name)
 return search_series_name_private(key, n->left_child);

// Otherwise, search the right subtree
 else
 return search_series_name_private(key, n->right_child);
}

// DEBUG statement that prints an error message when the entry is not found
 #ifdef DEBUG
 std::cout << ``[ERROR][TREE::search_series_name_priv] Entry was not found in
database...'' << std::endl;
 #endif

 return NULL;
}

// This function searches for all the TV series present in the tree
// that a particular actor has acted in and adds them to the shows_of_actor
vector.
void Tree::get_series_vector_from_actor(std::vector<std::string>
&shows_of_actor, tree_node* entry, std::string actor)
{
 // If the current node is not NULL
 if(entry != NULL)
 {

// If the current node contains the actor we are looking for
 if(search_actor_list_of_node(entry, actor))

// Add the TV series to the shows_of_actor vector
 shows_of_actor.push_back(entry->node_data.series_name);

// Recursively search the left subtree for TV series that the actor has acted
in
 get_series_vector_from_actor(shows_of_actor, entry->left_child, actor);

// Recursively search the right subtree for TV series that the actor has acted
in
 get_series_vector_from_actor(shows_of_actor, entry->right_child, actor);
 }
}

// This function searches for all the TV series present in the tree that were
aired
// between a particular range of years and adds them to the series_of_range
vector.
void Tree::get_series_vector_from_range(std::vector<std::string>
&series_of_range, tree_node* entry, int high, int low)
{
```

```
// If the current node is not NULL
 if(entry != NULL)
 {

// If the current node's airing year falls within the given range
 if(
 (entry->node_data.year_start <= high && entry->node_data.year_end >= low)||
 (entry->node_data.year_end >= low && entry->node_data.year_end <= high) ||
 (entry->node_data.year_start <= low && entry->node_data.year_end >= high)||
 (entry->node_data.year_start <= low && entry->node_data.year_end >= high)
 )

// Add the TV series to the series_of_range vector
 series_of_range.push_back(entry->node_data.series_name);

// Recursively search the left subtree for TV series that aired in the given
year range
 get_series_vector_from_range(series_of_range, entry->left_child, high, low);

// Recursively search the right subtree for TV series that aired in the given
year range
 get_series_vector_from_range(series_of_range, entry->right_child, high, low);
 }
}
// Constructor for the Tree class. It initializes the root_node to NULL and
// creates a vector of movie_data objects. Then, it creates a tree based on the
// data in the vector.
Tree::Tree()
{
 root_node = NULL;
 std::vector<movie_data> data_vector = create_movie_data_vector();
 create_tree(data_vector);
}

// Destructor for the Tree class. It deallocates the memory used by the root
node.
Tree::~Tree()
{
 delete root_node;
}

// This function creates a new tree_node object with the given movie_data
object.
// It returns a pointer to the newly created node.
tree_node* Tree::create_node(movie_data dat)
{
 #ifdef DEBUG
 std::cout << ``[DEBUG][TREE::create_node] Entering'' << std::endl;
 #endif

 tree_node* new_node = new tree_node;
 new_node->left_child = NULL;
 new_node->right_child = NULL;
 new_node->node_data = dat;

 #ifdef DEBUG
```

```
 std::cout << ``[DEBUG][TREE::create_node] Exiting'' << std::endl;
 #endif

 return new_node;
}

// This function inserts a node into the tree. It is called by the public
insert_node
// function and recursively searches the tree to find the correct position for
the new node.
void Tree::insert_node_public(tree_node* insertable_node)
{
 #ifdef DEBUG
 std::cout << ``[DEBUG][TREE::insert_node_public] Entering'' << std::endl;
 #endif

// If the root_node is not NULL, recursively search the tree for the correct
position for the new node.
 if(root_node != NULL){
 insert_node_private(root_node, insertable_node);
 }else
 root_node = insertable_node;

 #ifdef DEBUG
 std::cout << ``[DEBUG][TREE::insert_node_public] Exiting'' << std::endl;
 #endif
}

// This function creates a tree using the given vector of movie_data objects.
// It creates a new tree_node for each movie_data object and inserts it into
the tree.
void Tree::create_tree(std::vector<movie_data> data_vector)
{
 #ifdef DEBUG
 std::cout << ``[DEBUG][TREE::create_tree] Entering'' << std::endl;
 #endif

 for(int i=0; i < data_vector.size(); i++)
 {
 #ifdef DEBUG
 std::cout << ``[DEBUG][TREE::create_tree] Creating node ``<<
data_vector[i].series_name << std::endl;
 #endif

// Create a new node for the current movie_data object
 tree_node* new_node = create_node(data_vector[i]);

// Insert the new node into the tree
 insert_node_public(new_node);
 }

 #ifdef DEBUG
 std::cout << ``[DEBUG][TREE::create_tree] Exiting'' << std::endl;
 #endif
}
```

```
// This function searches for a particular TV series name in the tree and
// returns the node that contains that TV series.
tree_node* Tree::search_series_name_public(std::string series_name)
{

// Call the private search_series_name function with the root node of the tree.
// This is the public interface for the search_series_name function.
 return search_series_name_private(series_name, root_node);
}

// This function searches for a TV series in the tree and displays all the
actors who have acted in it.
void Tree::get_actor_from_series(std::string series_title)
{

// Search for the node containing the TV series
 tree_node* target_node = search_series_name_public(series_title);

// If the node is found
 if(target_node != NULL)

// Display the list of actors who have acted in the TV series
 display_actors(target_node);
 else

// Display an error message if the TV series is not found in the database
 std::cout << ``[OUTPUT]'' << series_title << '' not found in database.'' <<
std::endl;
}

// This function prints out all the TV series present in the tree
// that a particular actor has acted in.
void Tree::get_series_from_actor(std::string actor)
{

// Create a vector to store the TV series of the actor
 std::vector<std::string> series_of_actor;

// Get the TV series of the actor by calling the helper function
 get_series_vector_from_actor(series_of_actor, root_node, actor);

// Print the actor's␣name
 std::cout << ``[OUTPUT] Actor: '' << actor << ``\n";

// Print the TV series the actor has acted in
 std::cout << ``[OUTPUT] Seen in: ``;
 for(int k = 0; k < series_of_actor.size(); k++)
 {
 std::cout << series_of_actor[k];

// If there are more TV series, print a comma to separate them
 if(k != series_of_actor.size()-1)
 std::cout << ``, ``;
 }

// Print a newline character at the end
```

```
 std::cout << ``\n";
}

// This function searches for all the TV series present in the tree
// that fall within a given range of years and prints them to the console.
void Tree::get_series_within_range(int high, int low){

// Create a vector to hold the TV series that fall within the given range
 std::vector<std::string> series_of_range;

// Call the get_series_vector_from_range function to populate the
series_of_range vector
 get_series_vector_from_range(series_of_range, root_node, high, low);

// Print the TV series to the console
 std::cout << "[OUTPUT] Series from '' << high << ``-'' << low << ``:\n\t";
 for(int q = 0; q < series_of_range.size(); q++)
 {
 std::cout << series_of_range[q];
 if(q != series_of_range.size()-1)
 std::cout << ", ";
 }
 std::cout << ``\n";
}

// This function searches for a particular actor in the actor vector of a given
node.
// It returns true if the actor is present in the vector, and false otherwise.
bool Tree::search_actor_list_of_node(tree_node* target, std::string actor)
{

// Loop through the actor vector of the node
 for(int j = 0; j < target->node_data.actor_vector.size(); j++)
 {

// If the current actor matches the one we are searching for, return true
 if(target->node_data.actor_vector[j] == actor)
 return true;
 }

// If the actor is not found, return false
 return false;
}


// This function displays the names of actors associated with a particular TV
series
// stored in the node pointed to by displayable_node.
void Tree::display_actors(tree_node* displayable_node){


// Iterate through the vector of actors and display each actor's name
 for(int i = 0; i < displayable_node->node_data.actor_vector.size(); i++)
 std::cout << displayable_node->node_data.actor_vector[i] << ``, ``;
```

```
// Move to the next line for better readability
 std::cout << ``\n";
}

// This function displays the data present in the tree to the user.
// It calls a private function, display_data_private(), with the root_node as
the argument.
void Tree::display_data_public(){

// Call the private function, display_data_private(), with the root_node as the
argument.
 display_data_private(root_node);
}

// This function displays the TV series name of the current node.
// It can also display additional information about the series (currently
commented out).
void Tree::display(tree_node* displayable_node){

// Display the name of the TV series.
 std::cout <<
"[OUTPUT][TREE::DISPLAY] Series Name:"<<
 displayable_node->node_data.series_name<<``\n";
}
#endif
```

4.3. file_reader.h

```
/* file_reader.h
 *
 * CS 121.Bolden.......Compiler Version: 12.0.5...........Andreas Neacsu
 * 4/26/2023, ............M1 Macintosh.......neac9115@vandals.uidaho.edu
 *
 *
 * Header file for file_reader.cpp includes function / structure declarations
 *-------------------------------------------------------------------------
 */
#ifndef FILE_READER_H
#define FILE_READER_H
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <vector>

// This struct represents the data associated with a TV series.
struct movie_data
{
 std::string series_name;
 int year_start;
 int year_end;
 std::string series_category;
 std::string series_URL;
 std::vector<std::string> actor_vector;
 int index;
};
```

```
// This function creates a vector of movie_data structs by reading data from a
file.
std::vector<movie_data> create_movie_data_vector();

// This function extracts the name of a TV series from a given string.
void get_series_name(char [], char []);

// This function returns the index of a given character in a given string.
int index_of(char [], char);

// This function extracts the start and end years of a TV series from a given
string.
void get_years(char [], int& , int& );

// This function extracts a substring from a given string.
void get_subString(char [], int , int , char []);

// This function removes any blank spaces from a given string.
void remove_blanks( char []);
#endif
```

4.4. file_reader.cpp

```
/* file_reader.cpp
 *
 * CS 121.Bolden.......Compiler Version: 12.0.5...........Andreas Neacsu
 * 4/26/2023, ............M1 Macintosh.......neac9115@vandals.uidaho.edu
 *
 * This code defines the implementation of the file_reader functions declared
 * in the "file_reader.h" header file.
 *-------------------------------------------------------------------------
 */
#ifndef FILE_READER_CPP
#define FILE_READER_CPP
#include ‘‘file_reader.h"
//#define DEBUG

/*
This code defines a function named create_movie_data_vector(),
which returns a vector of movie_data objects.
The function reads data from a file named tv_DB.txt located in the ../data/
directory.
If the file cannot be opened, an error message is printed and the program
exits.
The file is read line-by-line using the getline() function and parsed to
extract various pieces of information
such as the series name, years the series was aired, series category, series
URL, and actor names.
After the data is parsed, it is stored in a movie_data object and added to the
vector using the push_back() function.
The n_series variable is incremented each time a new series is added to the
vector.
The code also includes some preprocessor directives #ifdef DEBUG which
indicates
that some lines are included only when the DEBUG flag is defined.
The code also defines several helper functions, such as get_series_name(),
get_years(), index_of(),
```

```
get_subString(), and remove_blanks(). These functions help to extract the
desired information
from each line of the input file.
Finally, the function returns the vector of movie_data objects and closes the
input file.
*/

// This function reads a file containing TV series data and returns a vector of
movie_data objects
std::vector<movie_data> create_movie_data_vector()
{

// Initialize variables
 std::vector<movie_data> movie_data_vector;
 std::ifstream file_in("../data/tv_DB.txt",std::ios::in);

// Check if file can be opened
 if(!file_in){
 std::cout << ``[ERROR][FILE_READER :: read_tv_file()] Unable to Read
File"<<std::endl;
 exit(-1);
 }

// Initialize variables for parsing each line of the file
 const int MAX_LINE = 128;
 char line[MAX_LINE];
 char series_name[MAX_LINE];
 char series_category[MAX_LINE/2];
 char series_URL[MAX_LINE];
 char actor_name[MAX_LINE/2];
 int y_start, y_end;
 int n_series = 0;

// Loop through each line of the file and parse the data
 while(file_in.getline(line, MAX_LINE))
 {
// Create a new movie_data object to store the parsed data
 movie_data new_movie_data;

// If the line is empty, skip to the next line
 if(strlen(line) == 0)
 continue;

// Parse the series name and years from the line
 get_series_name(line,series_name);
 get_years(line, y_start, y_end);

// Parse the series category and URL from the next two lines
 file_in.getline(series_category, MAX_LINE/2);
 file_in.getline(series_URL, MAX_LINE);

// Assign the parsed data to the movie_data object
 new_movie_data.series_name = series_name;
 new_movie_data.year_start = y_start;
 new_movie_data.year_end = y_end;
 new_movie_data.series_category = series_category;
```

```
 new_movie_data.series_URL = series_URL;

// Parse the actor names from the remaining lines
 file_in.getline( line, MAX_LINE/2 );
 int n = -1;
 while( strlen(line) > 0 ) {
 new_movie_data.actor_vector.push_back(line);
 file_in.getline( line, MAX_LINE/2 );
 n++;
}

// Assign an index to the movie_data object and add it to the vector
 new_movie_data.index = n_series;
 movie_data_vector.push_back(new_movie_data);

 n_series++;
 }

// Close the file and return the vector of movie_data objects
 file_in.close();
 return movie_data_vector;
}

// This function extracts the series name from a line of text
void get_series_name(char line[], char series_name[]){
 int start_year;
 start_year = index_of(line, '(');
 get_subString(line, 0, start_year-1, series_name);
}

void get_years( char line[], int & y_start, int & y_end )
{
 char tmp_string[8];
 char year_string[16];
 int year_start, year_end;

 year_start = index_of( line, '(' );
 year_end = index_of( line, ')' );

 get_subString( line, year_start+1, year_end-year_start-1, year_string);

 get_subString( year_string, 0, 4, tmp_string );
 y_start = atoi( tmp_string );

 get_subString( year_string, 5, 4, tmp_string ); // 7? not '-'!
 y_end = atoi( tmp_string );
}

// This function returns the index of the first occurrence of character 'c' in
string 's'.
int index_of(char s[], char c) {

// Initialize the index variable to 0
 int i = 0;

// While the current character is not a null terminator or 'c'
```

```
 while (s[i] != '\0' && s[i] != c) {

// Increment the index
 i++;
 }

// Return the index of the first occurrence of 'c' or the length of the string
if 'c' is not found.
 return i;
}

// This function extracts a substring from 's' and stores it in 'res', starting
at index 'start' and ending at index 'start+end-1'.
void get_subString(char s[], int start, int end, char res[]) {
 // Declare a counter variable
 int i;
// Declare and initialize the index of the result substring to 0
 int i_res = 0;

 for (i = start; i < start + end; i++) { // Iterate through 's' from index
'start' to 'start+end-1'
 res[i_res++] = s[i]; // Copy each character of the substring into 'res'
 }

// Add a null terminator to the end of the result substring
 res[i_res] = '\0';

// Remove any trailing whitespace from the result substring
 remove_blanks(res);
}

// This helper function removes any trailing whitespace (spaces or tabs) from a
string.
void remove_blanks(char s[]) {

// Calculate the length of the string
 int s_len = strlen(s);

// Iterate through the string from the end
 for (int i = s_len; i >= 0; i--) {

// If the current character is alphabetic, stop iterating
 if (isalpha(s[i])) {
 break;
 }

// If the current character is a space, replace it with a null terminator
 if (s[i] == ' ') { // If the current character is a space, replace it with a
null terminator
 s[i] = '\0';
 }
 }
}
#endif
```

4.5. main.h

```
/* main.h
 *
 * CS 121.Bolden.......Compiler Version: 12.0.5...........Andreas Neacsu
 * 4/26/2023, ............M1 Macintosh.......neac9115@vandals.uidaho.edu
 *
 * This code declares and defines a few functions that are usful for the user
 * to help displaying and traverse the binary tree.
 *-------------------------------------------------------------------------
 */
/*
This code defines several functions that create an interface for the user to
interact with the Tree database.
The interface includes options for displaying the entire database, displaying
all actors for a particular series,
displaying all series for a particular actor, displaying all series within a
specified range, and exiting the program.
Each option corresponds to a specific function call for the corresponding
operation on the Tree object.
*/
#ifndef MAIN_H
#define MAIN_H

#include <iostream>
#include "../tree/tree.h"

// This function displays the entire database of TV series in the tree.
// It calls the public display_data_public() function of the Tree class to
accomplish this.
void display_database(Tree tree)
{

// Call the public display_data_public() function of the Tree class to display
the database.
 tree.display_data_public();
}

// This function prompts the user to enter the name of an actor and
// then calls the get_series_from_actor function of the provided tree object
// to find all the TV series that the actor has acted in.
void get_series_of_actor(Tree tree){

// Declare a variable to store the user's␣input␣actor␣name
 std::string user_actor;
 std::cout << ``[INPUT] Enter Actor Name \n[INPUT] > ``;

// Get the user's␣input␣for␣actor␣name␣and␣store␣it␣in␣user_actor
 std::getline(std::cin, user_actor);

// Clear the input buffer
 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

// Call the get_series_from_actor function of the provided tree object
// to find all the TV series that the actor has acted in.
 tree.get_series_from_actor(user_actor);
```

```
}

// This function gets the actors who have acted in a particular TV series.
void get_actors(Tree tree){

// Ask the user to input the TV series name.
 std::string user_series;
 std::cout << ``[INPUT] Enter Series Name \n[INPUT] > ``;
 std::getline(std::cin, user_series);

// Clear any remaining input in the buffer.
 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

// Call the Tree class's␣get_actor_from_series␣function␣to␣get␣the␣actors.
 tree.get_actor_from_series(user_series);
}

// This function prompts the user to input a range of years and then
// calls the get_series_within_range function of the Tree class to retrieve
// all the TV series that aired within that range.
void get_series_of_range(Tree tree){
int user_range_start, user_range_end;

// Prompt the user to input the start of the range
 std::cout << ``[INPUT] Enter Range Start\n[INPUT] > ``;
 std::cin >> user_range_start;
 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
 // Clear input buffer

// Prompt the user to input the end of the range
 std::cout << ``[INPUT] Enter Range End\n[INPUT] > ``;
 std::cin >> user_range_end;
 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
// Clear input buffer

// Call the get_series_within_range function of the Tree class to retrieve all
the
// TV series that aired within the range specified by the user.
 tree.get_series_within_range(user_range_end, user_range_start);
}

// This function acts as an interface for the user to interact with the
database stored in the tree.
void database_interface(Tree tree){
 int selectoriovario = 0;

// Continue running the interface until the user chooses to exit.
 while(selectoriovario!=4){

// Display the options for the user to select.
 std::cout <<
 ``[OUTPUT][MAIN::database_interface]\n\t[0] Display Database\n\t[1] Display
all actors of a series\n\t[2] Display all series of a actor\n\t[3] Display all
series within a range\n\t[4] Exit\n[INPUT] > ``;

// Get user input for the selected option.
```

```
 std::cin >> selectoriovario;

// Clear input buffer to prevent any issues.
 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

// Perform the action corresponding to the selected option.
 switch (selectoriovario)
 {
 case 0:
// Display the entire database.
   display_database(tree);
   break;
 case 1:
// Display all actors of a particular TV series.
   get_actors(tree);
   break;
 case 2:
// Display all TV series that a particular actor has acted in.
   get_series_of_actor(tree);
   break;
 case 3:
// Display all TV series that fall within a given range.
   get_series_of_range(tree);
   break;
 default:
// Exit the interface.
   break;
 }
 }
}
#endif
```

4.6. main.cpp

```
/* main.cpp
 *
 * CS 121.Bolden.......Compiler Version: 12.0.5...........Andreas Neacsu
 * 4/26/2023, ............M1 Macintosh.......neac9115@vandals.uidaho.edu
 *-------------------------------------------------------------------
This code defines the main function for the program, which creates a Tree
object named ``database'' and passes it to the database_interface function.
The database_interface function allows the user to interact with the database
and perform various operations such as searching for data and displaying
the data in the tree
*/
#ifndef MAIN_CPP
#define MAIN_CPP
#include <iostream>

// The following line includes the header file for the main function in the
program.
#include ``main.h"

// The main function is the entry point for the program and initializes a Tree
object
// named database. It then passes the database object to the database_interface
function
```

```
// which allows the user to interact with the database.
int main(){
 Tree database;
 database_interface(database);
 return 0;
}
#endif
```

5. Output

5.1. Display Database (Titles Only)

```
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 0
[OUTPUT][TREE::DISPLAY] Series Name: 3rd Rock from the Sun
[OUTPUT][TREE::DISPLAY] Series Name: Alfred Hitchcock Presents
[OUTPUT][TREE::DISPLAY] Series Name: All in the Family
[OUTPUT][TREE::DISPLAY] Series Name: American Dad!
[OUTPUT][TREE::DISPLAY] Series Name: Animaniacs
[OUTPUT][TREE::DISPLAY] Series Name: Babylon
[OUTPUT][TREE::DISPLAY] Series Name: Banacek
[OUTPUT][TREE::DISPLAY] Series Name: Batman
[OUTPUT][TREE::DISPLAY] Series Name: Battlestar Galactica
[OUTPUT][TREE::DISPLAY] Series Name: Benson
[OUTPUT][TREE::DISPLAY] Series Name: Bewitched
[OUTPUT][TREE::DISPLAY] Series Name: Burke's Law
[OUTPUT][TREE::DISPLAY] Series Name: CHiPs
[OUTPUT][TREE::DISPLAY] Series Name: Charlie's Angels
[OUTPUT][TREE::DISPLAY] Series Name: Chicago Hope
[OUTPUT][TREE::DISPLAY] Series Name: Coach
[OUTPUT][TREE::DISPLAY] Series Name: Criminal Minds
[OUTPUT][TREE::DISPLAY] Series Name: Dexter's Laboratory
[OUTPUT][TREE::DISPLAY] Series Name: ER
[OUTPUT][TREE::DISPLAY] Series Name: Evening Shade
[OUTPUT][TREE::DISPLAY] Series Name: F Troop
[OUTPUT][TREE::DISPLAY] Series Name: Family Ties
[OUTPUT][TREE::DISPLAY] Series Name: Futurama
[OUTPUT][TREE::DISPLAY] Series Name: Gidget
[OUTPUT][TREE::DISPLAY] Series Name: Gilligan's Island
[OUTPUT][TREE::DISPLAY] Series Name: Gomer Pyle: USMC
[OUTPUT][TREE::DISPLAY] Series Name: Happy Days
[OUTPUT][TREE::DISPLAY] Series Name: Hawaii Five-0
[OUTPUT][TREE::DISPLAY] Series Name: Herman's Head
[OUTPUT][TREE::DISPLAY] Series Name: Hogan's Heroes
[OUTPUT][TREE::DISPLAY] Series Name: I Dream of Jeannie
[OUTPUT][TREE::DISPLAY] Series Name: I Love Lucy
[OUTPUT][TREE::DISPLAY] Series Name: Ironside
```

```
[OUTPUT][TREE::DISPLAY] Series Name: JAG
[OUTPUT][TREE::DISPLAY] Series Name: Jake and the Fatman
[OUTPUT][TREE::DISPLAY] Series Name: Kojak
[OUTPUT][TREE::DISPLAY] Series Name: Kung Fu: The Legend Continues
[OUTPUT][TREE::DISPLAY] Series Name: Lassie
[OUTPUT][TREE::DISPLAY] Series Name: Law & Order
[OUTPUT][TREE::DISPLAY] Series Name: Leave It to Beaver
[OUTPUT][TREE::DISPLAY] Series Name: Little House on the Prairie
[OUTPUT][TREE::DISPLAY] Series Name: Lost
[OUTPUT][TREE::DISPLAY] Series Name: Lost in Space
[OUTPUT][TREE::DISPLAY] Series Name: M*A*S*H
[OUTPUT][TREE::DISPLAY] Series Name: MacGyver
[OUTPUT][TREE::DISPLAY] Series Name: Make Room for Daddy
[OUTPUT][TREE::DISPLAY] Series Name: Mannix
[OUTPUT][TREE::DISPLAY] Series Name: Married with Children
[OUTPUT][TREE::DISPLAY] Series Name: Mary Tyler Moore
[OUTPUT][TREE::DISPLAY] Series Name: Matlock
[OUTPUT][TREE::DISPLAY] Series Name: McCloud
[OUTPUT][TREE::DISPLAY] Series Name: McHale's Navy
[OUTPUT][TREE::DISPLAY] Series Name: Mister Ed
[OUTPUT][TREE::DISPLAY] Series Name: Mod Squad
[OUTPUT][TREE::DISPLAY] Series Name: Mork & Mindy
[OUTPUT][TREE::DISPLAY] Series Name: Mr. Lucky
[OUTPUT][TREE::DISPLAY] Series Name: Murder, She Wrote
[OUTPUT][TREE::DISPLAY] Series Name: My Three Sons
[OUTPUT][TREE::DISPLAY] Series Name: NCIS
[OUTPUT][TREE::DISPLAY] Series Name: NCIS: Los Angeles
[OUTPUT][TREE::DISPLAY] Series Name: NCIS: New Orleans
[OUTPUT][TREE::DISPLAY] Series Name: Newhart
[OUTPUT][TREE::DISPLAY] Series Name: Night Court
[OUTPUT][TREE::DISPLAY] Series Name: Northern Exposure
[OUTPUT][TREE::DISPLAY] Series Name: Quantum Leap
[OUTPUT][TREE::DISPLAY] Series Name: Rawhide
[OUTPUT][TREE::DISPLAY] Series Name: Riptide
[OUTPUT][TREE::DISPLAY] Series Name: Room
[OUTPUT][TREE::DISPLAY] Series Name: Scarecrow and Mrs. King
[OUTPUT][TREE::DISPLAY] Series Name: Seinfeld
[OUTPUT][TREE::DISPLAY] Series Name: St. Elsewhere
[OUTPUT][TREE::DISPLAY] Series Name: Star Trek
[OUTPUT][TREE::DISPLAY] Series Name: Star Trek: Deep Space Nine
[OUTPUT][TREE::DISPLAY] Series Name: Star Trek: Enterprise
[OUTPUT][TREE::DISPLAY] Series Name: Star Trek: The Next Generation
[OUTPUT][TREE::DISPLAY] Series Name: Star Trek: Voyager
[OUTPUT][TREE::DISPLAY] Series Name: Taxi
[OUTPUT][TREE::DISPLAY] Series Name: The A-Team
[OUTPUT][TREE::DISPLAY] Series Name: The Adventures of Ozzie & Harriet
[OUTPUT][TREE::DISPLAY] Series Name: The Andy Griffith Show
[OUTPUT][TREE::DISPLAY] Series Name: The Beverly Hillbillies
[OUTPUT][TREE::DISPLAY] Series Name: The Big Valley
[OUTPUT][TREE::DISPLAY] Series Name: The Bob Newhart Show
[OUTPUT][TREE::DISPLAY] Series Name: The Bullwinkle Show
[OUTPUT][TREE::DISPLAY] Series Name: The Carol Burnett Show
[OUTPUT][TREE::DISPLAY] Series Name: The Cosby Show
[OUTPUT][TREE::DISPLAY] Series Name: The Fall Guy
[OUTPUT][TREE::DISPLAY] Series Name: The Flying Nun
[OUTPUT][TREE::DISPLAY] Series Name: The Fugitive
```

```
[OUTPUT][TREE::DISPLAY] Series Name: The Honeymooners
[OUTPUT][TREE::DISPLAY] Series Name: The Invaders
[OUTPUT][TREE::DISPLAY] Series Name: The Jack Benny Program
[OUTPUT][TREE::DISPLAY] Series Name: The Jeffersons
[OUTPUT][TREE::DISPLAY] Series Name: The Lucy Show
[OUTPUT][TREE::DISPLAY] Series Name: The Man from U.N.C.L.E.
[OUTPUT][TREE::DISPLAY] Series Name: The Many Loves of Dobie Gillis
[OUTPUT][TREE::DISPLAY] Series Name: The Odd Couple
[OUTPUT][TREE::DISPLAY] Series Name: The Office
[OUTPUT][TREE::DISPLAY] Series Name: The Phil Silvers Show
[OUTPUT][TREE::DISPLAY] Series Name: The Prisoner
[OUTPUT][TREE::DISPLAY] Series Name: The Saint
[OUTPUT][TREE::DISPLAY] Series Name: The Simpsons
[OUTPUT][TREE::DISPLAY] Series Name: The Six Million Dollar Man
[OUTPUT][TREE::DISPLAY] Series Name: The Streets of San Francisco
[OUTPUT][TREE::DISPLAY] Series Name: The Twilight Zone
[OUTPUT][TREE::DISPLAY] Series Name: The Wild Wild West
[OUTPUT][TREE::DISPLAY] Series Name: The X-Files
[OUTPUT][TREE::DISPLAY] Series Name: Topper
[OUTPUT][TREE::DISPLAY] Series Name: Voyage to the Bottom of the Sea
[OUTPUT][TREE::DISPLAY] Series Name: WKRP in Cincinnati
[OUTPUT][TREE::DISPLAY] Series Name: Walker, Texas Ranger
[OUTPUT][TREE::DISPLAY] Series Name: Wonder Woman
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 4
```

5.2. Display all Actors of a Given Show

```
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 1
[INPUT] Enter Series Name
[INPUT] > NCIS

Michael Weatherly, Pauley Perrette, David McCallum, Mark Harmon, Sean Murray,
Cote de Pablo, Brian Dietzen, Rocky Carroll, Lauren Holly, Sasha Alexander, Joe
Spano
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 1
[INPUT] Enter Series Name
[INPUT] > McHale's Navy
```

```
Ernest Borgnine, Joe Flynn, Tim Conway, Carl Ballantine , Gary Vinson, Billy
Sands, Edson Stroll, John Wright, Yoshio Yoda, Bob Hastings, Gavin MacLeod
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 1
[INPUT] Enter Series Name
[INPUT] > The Prisoner

Patrick McGoohan, Angelo Muscat, Peter Swanwick, Leo McKern
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 1
[INPUT] Enter Series Name
[INPUT] > The Office

Rainn Wilson, John Krasinski, Jenna Fischer, Leslie David Baker, Brian
Baumgartner, Angela Kinsey, Phyllis Smith, Kate Flannery, Creed Bratton , Oscar
Nunez, B.J. Novak , Mindy Kaling
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 1
[INPUT] Enter Series Name
[INPUT] > Matlock

Andy Griffith, Nancy Stafford, Julie Sommars, Clarence Gilyard Jr., Kene
Holliday
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 1
[INPUT] Enter Series Name
[INPUT] > Star Trek: Voyager

Kate Mulgrew , Robert Beltran, Roxann Dawson, Robert Duncan McNeill, Ethan
Phillips, Robert Picardo, Tim Russ, Garrett Wang, Tarik Ergin, Majel Barrett,
Jeri Ryan
```

5.3. Display All Shows of a Given Actor

```
[OUTPUT][MAIN::database_interface]
        [0] Display Database
```

```
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 2
[INPUT] Enter Actor Name
[INPUT] > Bill Daily

[OUTPUT] Actor: Bill Daily
[OUTPUT] Seen in: I Dream of Jeannie, The Bob Newhart Show
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 2
[INPUT] Enter Actor Name
[INPUT] > Dana Elcar

[OUTPUT] Actor: Dana Elcar
[OUTPUT] Seen in: MacGyver
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 2
[INPUT] Enter Actor Name
[INPUT] > Andy Griffith

[OUTPUT] Actor: Andy Griffith
[OUTPUT] Seen in: Matlock, The Andy Griffith Show
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 2
[INPUT] Enter Actor Name
[INPUT] > Tress MacNeille

[OUTPUT] Actor: Tress MacNeille
[OUTPUT] Seen in: Futurama, Animaniacs, The Simpsons
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 2
[INPUT] Enter Actor Name
[INPUT] > Larry Storch
```

```
[OUTPUT] Actor: Larry Storch
[OUTPUT] Seen in: F Troop
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 2
[INPUT] Enter Actor Name
[INPUT] > Werner Klemperer

[OUTPUT] Actor: Werner Klemperer
[OUTPUT] Seen in: Hogan's Heroes
```

5.4. Display All Shows Released Between a Range

```
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 3
[INPUT] Enter Range Start
[INPUT] > 2005
[INPUT] Enter Range End
[INPUT] > 2015
[OUTPUT] Series from 2015-2005:
JAG, Futurama, American Dad!, Criminal Minds, ER, Hawaii Five-0, Law & Order,
Lost, NCIS, NCIS: Los Angeles, NCIS: New Orleans, Star Trek: Enterprise, The
Office, The Simpsons
[OUTPUT][MAIN::database_interface]
        [0] Display Database
        [1] Display all actors of a series
        [2] Display all series of a actor
        [3] Display all series within a range
        [4] Exit
[INPUT] > 3
[INPUT] Enter Range Start
[INPUT] > 1980
[INPUT] Enter Range End
[INPUT] > 1990
[OUTPUT] Series from 1990-1980:
Benson, CHiPs, Charlie's Angels, Coach, Evening Shade, Family Ties, Happy Days,
Jake and the Fatman, Matlock, MacGyver, M*A*S*H, Law & Order, Little House on
the Prairie, Married with Children, Northern Exposure, Newhart, Mork & Mindy,
Murder, She Wrote, Night Court, The A-Team, Quantum Leap, Scarecrow and Mrs.
King, Riptide, Seinfeld, St. Elsewhere, Star Trek: The Next Generation, Taxi,
The Fall Guy, The Cosby Show, The Jeffersons, The Simpsons, WKRP in Cincinnati
```

6. PROGRAMMING LOG

6.1. Day 1

→ 5 Minutes

Read the program requirements

→ 15 Minutes
Brain dump..
Sketch out basic program design

→ 30 Minutes
Begin tree.h

→ 30 minutes
Begin working on file_reader.cpp

6.2. Day 2

→ 25 Minutes
Finish file_reader.cpp, edited slightly to return vector of new struct called movie data

→ 45 Minutes
Planned the functions for my tree class and wrote the defenitions

6.3. Day 3

→ 45 Minutes
Began writing the functions that will be used to initialize the tree class

1. `Tree()`
2. `create_node()`
3. `insert_node_public()`
4. `insert_node_private()`
5. `create_tree()`

→ 30 Minutes
Fix an issue I encountered with `insert_node_private()` as a recursive function I needed to pass the entry tree node by reference, otherwise the root_node would never be not NULL.

→ 35 Minutes
Began planning the functions that would be used for displaying the data in the tree

1. `display()`
2. `display_data_public()`
3. `display_data_private()`
4. `display_actors()`

6.4. Day 4

→ 20 Minutes
Planning the solution to the second program requirement, displaying the actors of a given show.

→ 20 minutes

→ Implemented a solution to this problem with 3 functions. `get_actor_from_series()`, this function calls `search_series_name_public()` which returns a node pointer if the series given is found. We then display the actors of the returned node if it is not null.

→ 15 Minutes
Planning the solution to the third program requirement, displaying the shows that star the same actor.

→ 30 Minutes
Implementing the solution. Wrote a private method `get_series_vector_from_actor()` it takes a reference to a vector. The function traverses the tree recursively checking the actor list of each node with `search_actor_list_of_node()`if we find the desired actor within a node, we add the series name to the vector. We then display the vector.

→ 20 Minutes
Planning a solution to the final program requirement, displaying all series of a given date range.

→ 40 Minutes
Writing and testing the required functions. Using a similar solution to program req 3, traversing the tree, checking a condition, and returning a vector of the series that meet the given requirement. In this case, series in a given range.

6.5. Day 5

→ 60 Minutes
Wrote some functions inside main.cpp to create menu to interact with the functions i had written for the tree class previously.

→ 30 Minutes
Fixed an issue i encountered when searching/inserting into the tree. Was only checking the 0th index of the string.

→ 180 minutes
Spent this time commenting and writing documentation for the program

→ 120 minutes
Spent this time writing this writeup