Alex McNurlin

CS121

Programming assignment 4

4/20/16

# Program Classes and Methods

**Bst.h**: A Class for a binary search tree. Uses the movie.h class for each node. Items in the tree are sorted by name of the movie

* void add\_node( Movie \*node );
  + Adds a node to the Binary Search Tree
* void print\_actors( string title );
  + Lists all of the actors from a given move
* void print\_tree( Leaf &node );
  + Prints the tree using recursion
* void print\_shows\_by\_actor( Leaf &node, string actor );
  + Finds all shows with a given actor using recursion and prints them to the screen
* void print\_shows\_of\_decade( Leaf &node, int year );
  + Finds all show beginning in a given decade and prints them to the screen

**movie.h:** A class to store information on a tv show.

* void add\_title( string title );
  + Add a shows title.
* void add\_start\_year( int year );
  + Add the year a show was created.
* void add\_end\_year( int year );
  + Add the year a show ended.
* void add\_category( string category );
  + Add the category of a show.
* void add\_url( string url );
  + Add the url to the shows IMDB page.
* void add\_actor( string actor );
  + Add an actor that was in the show. This is stored in a linked list.
* bool has\_actor( string actor );
  + Returns true if the show has a given actor.
* bool started\_in\_decade( int year );
  + Returns true if the show started within 10 years after the given year.
* void print\_actors();
  + Prints all the actors in the show.
* string get\_title();
  + Returns the title of the show.

**ll.h:** A linked list class used to store the actors in the movie class. This class was re-used from previous assignments. Methods not used for this program are not listed.

* void add\_node( string n );
  + Adds a node to the linked list
* void print\_nodes();
  + Prints the contents of every node
* bool has\_node( string test\_node );

# Programming Log

Monday, Apr 18: 3 hrs

* 8:00pm – 11:00pm:
  + Edited linked list class to work with strings
  + Wrote Movie class

Tuesday, Apr 19: 7 hrs

* 1:30pm – 6:30pm
  + Wrote Bst class
  + Finish writing Movie class
* 8:00pm – 10:00

Total: 10 hrs

**Things Encountered/learned**

* Making a recursive class method was easier than I thought, although I had to break it into 2 functions, because I couldn't figure out how to pass the private root node of the Bst as a function argument.
* Manipulating c++ strings is far easier than working with c strings.
  + I tried to write the functions to use only c strings, but I found it too difficult to work with when the c++ strings had the built in functionality to handle what I wanted.

# Output

########################################################  
Script started on Tue 19 Apr 2016 09:27:49 PM PDT

alexmcnurlin@Laquisha:~/Google Drive/schoolStuff/cs121/hw4$ ./hw4

Shows in the database:

3rd Rock from the Sun

Alfred Hitchcock Presents

All in the Family

American Dad!

Animaniacs

Babylon 5

Banacek

Batman

Battlestar Galactica

Benson

Bewitched

Burke's Law

CHiPs

Charlie's Angels

Coach

Dexter's Laboratory

ER

Evening Shade

F Troop

Family Ties

Futurama

Gidget

Gilligan's Island

Gomer Pyle: USMC

Happy Days

Herman's Head

Hogan's Heroes

I Dream of Jeannie

I Love Lucy

Ironside

JAG

Jake and the Fatman

Kojak

Kung Fu: The Legend Continues

Lassie

Law & Order

Leave It to Beaver

Little House on the Prairie

Lost

Lost in Space

M\*A\*S\*H

MacGyver

MacGyver

Mannix

Married with Children

Mary Tyler Moore

Matlock

McCloud

McHale's Navy

Mister Ed

Mod Squad

Mork & Mindy

Mr. Lucky

Murder, She Wrote

My Three Sons

NCIS

Newhart

Night Court

Northern Exposure

Quantum Leap

Rawhide

Riptide

Room 222

Scarecrow and Mrs. King

Seinfeld

Star Trek

Star Trek: Deep Space Nine

Star Trek: The Next Generation

Taxi

The A-Team

The Adventures of Ozzie & Harriet

The Andy Griffith Show

The Beverly Hillbillies

The Big Valley

The Bob Newhart Show

The Bullwinkle Show

The Carol Burnett Show

The Cosby Show

The Fall Guy

The Flying Nun

The Fugitive

The Honeymooners

The Invaders

The Jack Benny Program

The Jeffersons

The Lucy Show

The Man from U.N.C.L.E.

The Many Loves of Dobie Gillis

The Phil Silvers Show

The Saint

The Simpsons

The Six Million Dollar Man

The Streets of San Francisco

The Twilight Zone

The Wild Wild West

The X-Files

Topper

Voyage to the Bottom of the Sea

WKRP in Cincinnati

Walker, Texas Ranger

Wonder Woman

Actors in "The Saint":

Justine Lord

Leslie Crawford

Ivor Dean

Roger Moore

Actors in "Benson":

Didi Conn

Ethan Phillips

Rene Auberjonois

Missy Gold

Inga Swenson

James Noble

Robert Guillaume

Actors in "Dexter's Laboratory":

Rob Paulsen

Kathryn Cressida

Allison Moore

Tom Kenny

Frank Welker

Christine Cavanaugh

Jeff Bennett

Kath Soucie

Actors in "Futurama":

Maurice LaMarche

Lauren Tom

Phil LaMarr

Tress MacNeille

John DiMaggio

Katey Sagal

Billy West

Actors in "Star Trek":

Majel Barrett

Walter Koenig

George Takei

Eddie Paskey

James Doohan

# Code – hw4.cpp

########################################################

// Alex McNurlin

// File: hw4.cpp

// Created: 4/18/16

#include <iostream>

#include <fstream>

#include "bst.h"

#include <cstdlib>

using namespace std;

int main() {

// Open file

// This is hardcoded just to make testing easier

char filename[32] = "tvDB.txt";

//cout << "Enter filename: ";

//cin >> filename;

ifstream in\_file(filename);

if ( in\_file.fail() ) {

cout << "File " << filename << " is invalid " << endl << "Exiting..." << endl;

exit(1);

}

// I thought the file contained info on movies when I started this assignment.

// Thus some of the names don't make as much sense (like 'Movie' instead of show)

Bst movie\_tree;

string movie\_title;

string start\_year;

string end\_year;

string category;

string url;

string actor;

// Reads in contents from file show by show

// An instance of the Movie class is created and each instance is added

// to the Binary Search Tree

while( !in\_file.eof() ) {

Movie \*new\_movie = new Movie();

// Read in title (Everything until '(' )

getline( in\_file, movie\_title, '(' );

new\_movie->add\_title( movie\_title.erase( movie\_title.size()-1, 1 ) );

// ^ This erases the last character, which

// is always a space.

// Read in starting year (from '(' until '-' )

getline( in\_file, start\_year, '-' ); //

new\_movie->add\_start\_year( stoi(start\_year) );

// Read in end year (From '-' until '(' )

getline( in\_file, end\_year, ')' );

new\_movie->add\_end\_year( stoi(end\_year) );

getline( in\_file, actor ); // There is always an extra space after the ')'

// This gets rid of that

// Read in category

getline( in\_file, category );

new\_movie->add\_category( category );

// Read in url

getline( in\_file, url );

new\_movie->add\_url( url );

// Read in remaining lines until empty line as actors

while( 1 ) {

actor = "";

getline( in\_file, actor );

new\_movie->add\_actor( actor );

if ( actor == "" ) {

break;

}

}

// Add the show to the tree

movie\_tree.add\_node( new\_movie );

}

// Print out the necessary information

movie\_tree.print\_tree();

cout << endl;

movie\_tree.print\_actors("The Saint");

movie\_tree.print\_actors("Benson");

movie\_tree.print\_actors("Dexter's Laboratory");

movie\_tree.print\_actors("Futurama");

movie\_tree.print\_actors("Star Trek");

cout << endl;

movie\_tree.print\_shows\_by\_actor("Nancy Cartwright");

movie\_tree.print\_shows\_by\_actor("Marcia Wallace");

movie\_tree.print\_shows\_by\_actor("Bill Daily");

movie\_tree.print\_shows\_by\_actor("Joe Penny");

movie\_tree.print\_shows\_by\_actor("Michael Jeter");

movie\_tree.print\_shows\_by\_actor("Madeleine Sherwood");

cout << endl;

movie\_tree.print\_shows\_of\_decade( 1975 );

movie\_tree.print\_shows\_of\_decade( 2000 );

return 0;

}

# Code – Bst.h

########################################################  
// Alex McNurlin

// File: bst.h

// Created: 4/18/16

#include "movie.h"

#include <iostream>

#ifndef BST\_H

#define BST\_H

class Bst {

public:

typedef struct leaf {

Movie \*value;

leaf \*right = NULL;

leaf \*left = NULL;

} Leaf;

Bst() {

root = NULL;

}

void add\_node( Movie \*node );

void print\_actors( string title );

void print\_tree();

void print\_tree( Leaf &node );

void print\_shows\_by\_actor( string actor );

void print\_shows\_by\_actor( Leaf &node, string actor );

void print\_shows\_of\_decade( int year );

void print\_shows\_of\_decade( Leaf &node, int year );

private:

Leaf \*root;

};

#endif

# Code – Bst.cpp

########################################################

// Alex McNurlin

// File: bst.cpp

// Created: 4/18/16

#include "bst.h"

#include <iostream>

using namespace std;

void Bst::add\_node( Movie \*node ) {

if ( root == NULL ) { // Empty case

root = new Leaf();

root->value = node;

return;

}

Leaf \*p = root;

while ( 1 ) { // Traverse the tree and insert node

if ( node->get\_title() <= p->value->get\_title() ) { // Check left side

if ( p->left == NULL ) { // Node insertion

p->left = new Leaf();

p->left->value = node;

return;

} else

p = p->left;

} else if ( node->get\_title() > p->value->get\_title() ) {// Check right side

if ( p->right == NULL ) { // Node insertion

p->right = new Leaf();

p->right->value = node;

return;

} else

p = p->right;

}

}

}

void Bst::print\_actors( string title ) {

if ( root == NULL ) { // Empty case

cout << "Title \"" << title << "\" not found" << endl;

return;

}

Leaf \*p = root;

while ( 1 ) { // Traverse the tree and find title

if ( title <= p->value->get\_title() ) { // Check left branch

if ( p->value->get\_title() == title ) {// Check if the title matches

p->value->print\_actors(); // Print results

return;

} else if ( p->left == NULL ) { // If the title doesn't match and

// there's nowhere left to check

cout << "Title \"" << title << "\" not found" << endl;

return;

} else { // Check next interation

p = p->left;

}

} else if ( title > p->value->get\_title() ) {// right branch

if ( p->right == NULL ) { // If the title doesn't match and // there's nowhere left to check

cout << "Title \"" << title << "\" not found" << endl;

return;

} else { // Check next interation

p = p->right;

}

}

}

}

// Prints all shows of a given actor

// The first function is called first, which calls the second recursively

void Bst::print\_shows\_by\_actor( string actor ) {

Leaf \*p = root;

// Checks the left branch

cout << "Shows with actor \"" << actor << "\": " << endl;

if ( p->left != NULL ) {

print\_shows\_by\_actor( \*p->left, actor );

}

// Only prints if the actor is in the show

if ( p->value->has\_actor( actor ) ) {

cout << '\t' << p->value->get\_title() << endl;

}

// Checks the left branch

if ( p->right != NULL ) {

print\_shows\_by\_actor( \*p->right, actor );

}

}

void Bst::print\_shows\_by\_actor( Leaf &node, string actor ) {

Leaf \*p = new Leaf();

p = &node;

// Checks the left branch

if ( p->left != NULL ) {

print\_shows\_by\_actor( \*p->left, actor );

}

// Only prints if the actor is in the show

if ( p->value->has\_actor( actor ) ) {

cout << '\t' << p->value->get\_title() << endl;

}

// Checks the right branch

if ( p->right != NULL ) {

print\_shows\_by\_actor( \*p->right, actor );

}

}

void Bst::print\_shows\_of\_decade( int year ) {

Leaf \*p = root;

cout << "Shows starting in \"" << year << "-" << year+10 << "\": " << endl;

// Recursively check left branch

if ( p->left != NULL ) {

print\_shows\_of\_decade( \*p->left, year );

}

// Only print show if it's in the given decade

if ( p->value->started\_in\_decade( year ) ) {

cout << '\t' << p->value->get\_title() << endl;

}

// Recursively check right branch

if ( p->right != NULL ) {

print\_shows\_of\_decade( \*p->right, year );

}

}

void Bst::print\_shows\_of\_decade( Leaf &node, int year ) {

Leaf \*p = new Leaf();

p = &node;

// Recursively check left branch

if ( p->left != NULL ) {

print\_shows\_of\_decade( \*p->left, year );

}

// Only print show if it's in the given decade

if ( p->value->started\_in\_decade( year ) ) {

cout << '\t' << p->value->get\_title() << endl;

}

if ( p->right != NULL ) {

print\_shows\_of\_decade( \*p->right, year );

}

}

// The first function is called first, which calls the second recursively

void Bst::print\_tree() {

Leaf \*p = root;

cout << "Shows in the database: " << endl;

if ( p->left != NULL ) {

print\_tree( \*p->left );

}

cout << '\t' << p->value->get\_title() << endl;

if ( p->right != NULL ) {

print\_tree( \*p->right );

}

}

void Bst::print\_tree( Leaf &node ) {

Leaf \*p = new Leaf();

p = &node;

if ( p->left != NULL ) {

print\_tree( \*p->left );

}

cout << '\t' << p->value->get\_title() << endl;

if ( p->right != NULL ) {

print\_tree( \*p->right );

}

}

# Code – movie.h

########################################################

// Alex McNurlin

// File: movie.h

// Created: 4/18/16

#include "ll.h"

#include <string>

#ifndef MOVIE\_H

#define MOVIE\_H

// I thought the file contained info on movies when I started this assignment.

// Thus some of the names don't make as much sense (like 'Movie' instead of show)

class Movie {

public:

Movie() {};

void add\_title( string title );

void add\_start\_year( int year );

void add\_end\_year( int year );

void add\_category( string category );

void add\_url( string url );

void add\_actor( string actor );

bool has\_actor( string actor );

bool started\_in\_decade( int year );

void print\_actors();

string get\_title();

private:

string title;

int start\_year;

int end\_year;

string category;

string url;

Ll actors;

};

#endif

# Code – movie.cpp

########################################################

// Alex McNurlin

// File: movie.cpp

// Created: 4/18/16

#include "movie.h"

#include <string.h>

#include <iostream>

void Movie::add\_title( string new\_title ) {

title = new\_title;

}

void Movie::add\_start\_year( int year ) {

start\_year = year;

}

void Movie::add\_end\_year( int year ) {

end\_year = year;

}

void Movie::add\_category( string new\_category ) {

category = new\_category;

}

void Movie::add\_url( string new\_url ) {

url = new\_url;

}

void Movie::add\_actor( string actor ) {

actors.add\_node(actor);

}

bool Movie::has\_actor( string actor ) {

return actors.has\_node( actor );

}

bool Movie::started\_in\_decade( int year ) {

return ( start\_year <= year+10 && start\_year >= year );

}

void Movie::print\_actors() {

cout << "Actors in \"" << title << "\":";

actors.print\_nodes();

}

string Movie::get\_title() {

return title;

}

# Code – ll.h

########################################################

// Alex McNurlin

// File: ll.h

// Created: 2/25/16

// Edited: 4/15/16

// - Edited to use strings for hw4

#ifndef LL\_H

#define LL\_H

#include <string>

using namespace std;

// A class implementation of a linked list.

class Ll {

public:

Ll();

~Ll();

void add\_node( string n );

string delete\_head();

string get\_top();

int size();

void print\_nodes();

bool has\_node( string test\_node );

private:

struct node {

node \*next;

string value;

};

node \*head; // Points to the head of the list.

};

#endif

# Code – ll.cpp

########################################################

// Alex McNurlin

// File: ll.cpp

// Created: 2/25/16

// Edited: 4/15/16

// - Edited to use strings for hw4

#include "ll.h"

#include <cstddef>

#include <iostream>

#include <string>

using namespace std;

Ll::Ll() {

head = NULL;

}

Ll::~Ll() {

node \*p = head;

node \*q = p;

head = head->next;

while (q != NULL) {

p = q;

q = p->next;

delete p;

}

}

// Adding a single node to the head of the list.

void Ll::add\_node( string n ) {

node \*p;

p = new node();

p->value = n;

p->next = head;

head = p;

}

// Deletes the top node of the list. Does nothing if empty.

string Ll::delete\_head() {

if ( head != NULL ) {

string return\_value = head->value;

node \*p;

p = head;

head = head->next;

delete p;

return return\_value;

}

}

// Simply gets the value of the top node. Should return 0 if empty.

string Ll::get\_top() {

return head->value;

}

// Returns the size of the list. 0 if empty.

int Ll::size() {

int i = 0;

node \*p;

p = head;

while (p != NULL) {

i += 1;

p = p->next;

}

return i;

}

// Returns an array of the contents (type string)

void Ll::print\_nodes() {

node \*p = new node();

if ( head == NULL ) { // Do nothing

} else {

p = head;

cout << p->value << endl;

while ( p->next != NULL ) {

p = p->next;

cout << '\t' << p->value << endl;

}

}

}

bool Ll::has\_node( string test\_node ) {

node \*p = new node();

if ( head == NULL ) { // Do nothing

return false;

} else {

p = head;

if ( p->value == test\_node ) { // Test if the test\_node is at the head of list

return true;

}

while ( p->next != NULL ) { // Traverse the list

p = p->next;

if ( p->value == test\_node ) { // Test if the test\_node is in the list

return true;

}

}

return false; // Return false if no matches are found

}

}