CMP-5015Y Coursework 3 - Offline Movie Database in C++

100242165 (`dvd18scu`)

Wednesday 13th May, 2020 13:38

PDF prepared using LaTeX template v1.00 .

[✓] I agree that by submitting a PDF generated from this template I am confirming that I have checked the PDF and that it correctly represents my submission.

Contents

Movie.h

```
/*
By Robin Rai
V.1.0.0
Created on 05/03/2020
This file features the object declaration, all the headers, getters, and some
    overloaded operators
*/

#ifndef PROJECT_MOVIE_H
#define PROJECT_MOVIE_H

#include <string>
#include <iostream>

using namespace std;    //gav said it's okay to use this!

class Movie {
private:

    string title;
    string ageRating;
    //enum ageRating {PG13, APPROVED, R, PG, NOTRATED, G};
    //i'm not using enums as the ageRating string overloaded into an enum value
        would require perfect string input anyway.
    string genre;
    int year;
    int duration;
    float usrRating;


public:
    Movie(string title, int date, string ageRating, string filmGenre, int
        duration, int usrRating);

    Movie();

    ~Movie() {
        //cout << title << " Movie destroyed" << endl;
    };

    inline std::string getTitle() const {
        return this->title;
    }

    inline int getYear() const {
        return this->year;
    }

    inline string getAgeRating() const {
        return this->ageRating;
    }

    inline string getGenre() const {
        return this->genre;
    }

    inline int getDuration() const {
        return this->duration;
    }

    inline float getUsrRating() const {
        return this->usrRating;
    }

    friend inline ostream &operator<<(ostream &outputStream, const Movie &mov);

    friend istream &operator>>(istream &inputStream, Movie &mov);

    friend bool operator<(const Movie &mov1, const Movie &mov2);

    friend bool operator>(const Movie &mov1, const Movie &mov2);

    friend bool operator<=(const Movie &mov1, const Movie &mov2);

    friend bool operator>=(const Movie &mov1, const Movie &mov2);

    friend inline bool operator==(const Movie &mov1, const Movie &mov2);

    friend inline bool operator!=(const Movie &mov1, const Movie &mov2);
};

//lightweight so in headerfile and inlined
inline ostream &operator<<(ostream &outputStream, const Movie &mov) {
    //outputs a movie's data nicely when << operator is called on it
    outputStream << "\"" << mov.title << "\"," << mov.year << ",\"" << mov.
       ageRating << "\",\""
                 << mov.genre
                 << "\"," << mov.duration << "," << mov.usrRating << endl;
    return outputStream;
}

//same here for the inlining/headering
inline bool operator==(const Movie &mov1, const Movie &mov2) {
    //i'm not sure comparing by title also would be a great idea
    return mov1.year == mov2.year;
}

inline bool operator!=(const Movie &mov1, const Movie &mov2) {
    return mov1.year != mov2.year;
}

void movieTest();

//gotta be in header file to be inlined, gotta be friended to have access to
   variables
#endif //PROJECT_MOVIE_H
```

Movie.cpp

```cpp
/*
By Robin Rai
V.1.0.0
Created on 05/03/2020
This file features the Movie constructors, some overloaded operators, and a test
    function
*/

#include <string>
#include "Movie.h"
#include <vector>
#include <sstream>

using namespace std;    //I do not want to type std eight-hundred times

Movie::Movie(string title, int date, string ageRating, string genre, int duration
    , int usrRating) {
    this->title = title;
    this->year = date;
    this->ageRating = ageRating;
    this->duration = duration;
    this->usrRating = usrRating;
    this->genre = genre;

};

Movie::Movie() {
    this->title = "INVALID";
    this->ageRating = "INVALID";
    this->genre = "INVALID";
    this->year = 0000;
    this->duration = 0000;
    this->usrRating = 0000;
};

//big and chunky so not inline/in header file.
std::istream &operator>>(std::istream &inputStream, Movie &mov) {
    //you silly goose no const, it's literal only purpose is to edit mov
    string title, ageRating, genre;
    int year, duration, usrRating;

    char q; //empty char that represents where a quotation mark would be
    char c; //empty char that represents where a comma would be

    //if input is in the perfect layout, delimited by "s,
    if (inputStream
                >> q && getline(inputStream, title, '"')
                // " + title
                >> c >> year >> c
                // , year ,
                >> q && getline(inputStream, ageRating, '"') >> c
                // " + ageRating ,
                >> q && getline(inputStream, genre, '"')
                // " + genre
                >> c >> duration >> c
                // , duration ,
                >> usrRating)
        // usrRating

    {
        //set mov's parameters accordingly
        mov = Movie(title, year, ageRating, genre, duration, usrRating);
    } else {
        //otherwise fail with the flag, and just make a default Movie.
        inputStream.clear(ios_base::failbit);
        mov = Movie();
    }
    return inputStream;
}

//these aren't in the header file/inlined because they're quite chunky and have
   quite a few function calls.
bool operator<(const Movie &mov1, const Movie &mov2) {
    //logic for o1 < o2
    if (mov1.year == mov2.year) {
        //if the years are the same
        if (mov1.title.compare(mov2.title) == 0) {
            //if mov1's title is equal to mov2's
            return false;
        }
        if (mov1.title.compare(mov2.title) > 0) {
            //if mov1's title is bigger than mov2's
            return true;
        }
        //if mov1's title is smaller
        return false;
    } else {
        //man IDE's are advanced. If the years aren't the same return the right
           result.
        return mov1.year < mov2.year;
    }
}

bool operator>(const Movie &mov1, const Movie &mov2) {
    //logic for mov1 > mov2. Same as above so no comments
    if (mov1.year == mov2.year) {
        if (mov1.title.compare(mov2.title) == 0) {
            return false;
        }
        if (mov1.title.compare(mov2.title) < 0) {
            return true;
        }
        return false;
    } else {
        return mov1.year > mov2.year;
    }
}

bool operator<=(const Movie &mov1, const Movie &mov2) {
    //anything that's not mov1 > mov2
    if (mov1 > mov2) {
        return false;
    }
    return true;
}


bool operator>=(const Movie &mov1, const Movie &mov2) {
    //anything that's not mov1 < mov2
    if (mov1 < mov2) {
        return false;
    }
    return true;
}
```

```
void movieTest() {
    //string title, int date, string ageRating, string genre, int duration, int
       usrRating
    Movie test1("Test1", 2001, "PG", "Film-Noir", 144, 9.0);
    Movie test2("Test2", 2000, "PG", "Film-Noir", 144, 9.0);
    Movie test3("Test3", 2000, "PG", "Film-Noir", 144, 9.0);

    vector<Movie> test;
    test.push_back(test1);
    test.push_back(test2);
    test.push_back(test3);


    Movie test4;

    //"Seven Samurai",1954,"UNRATED","Action/Adventure/Drama",207,0
    string line = "\"Test4\",2000,\"PG\",\"Film-Noir\",144,9.0";
    std::istringstream iss(line);
    iss >> test4;

    test.push_back(test4);


    Movie test5;
    line = "999,\"Film-Noir\",pee is stored in the balls,9.0";
    std::istringstream iss2(line);
    iss2 >> test5;

    test.push_back(test5);
    Movie test6;
    test.push_back(test6);
    cout << "Movie.cpp test: " << endl;
    for (int i = 0; i < test.size(); i++) {
        cout << "Movie " << i + 1 << ": " <<
             endl;
        cout << test[i];
    }


    cout << "Movie 1 compared to Movie 2: " << endl;

    bool result1, result2, result3, result4, result5, result6;
    result1 = test1 < test2;
    result2 = test1 > test2;
    result3 = test1 == test2;
    result4 = test1 <= test2;
    result5 = test1 >= test2;
    result6 = test1 != test2;
    cout << "< " << result1 << endl;
    cout << "> " << result2 << endl;
    cout << "== " << result3 << endl;
    cout << "<= " << result4 << endl;
    cout << ">= " << result5 << endl;
    cout << "!= " << result6 << endl;


    cout << "Movie 2 compared to Movie 3: " << endl;


    result1 = test2 < test3;
    result2 = test2 > test3;
    result3 = test2 == test3;
    result4 = test2 <= test3;
    result5 = test2 >= test3;
    result6 = test2 != test3;
    cout << "< " << result1 << endl;
    cout << "> " << result2 << endl;
    cout << "== " << result3 << endl;
    cout << "<= " << result4 << endl;
    cout << ">= " << result5 << endl;
    cout << "!= " << result6 << endl;

    cout << "Movie 5 compared to Movie 6: " << endl;


    result1 = test5 < test6;
    result2 = test5 > test6;
    result3 = test5 == test6;
    result4 = test5 <= test6;
    result5 = test5 >= test6;
    result6 = test5 != test6;
    cout << "< " << result1 << endl;
    cout << "> " << result2 << endl;
    cout << "== " << result3 << endl;
    cout << "<= " << result4 << endl;
    cout << ">= " << result5 << endl;
    cout << "!= " << result6 << endl;

}
```

MovieDatabase.h

```
/*
By Robin Rai
V.1.0.0
Created on 05/03/2020
 This file features the declaration for MovieDatabase, some getters, and headers
*/

#include <vector>
#include <algorithm>
#include "Movie.h"

#ifndef PROJECT_MOVIEDATABASE_H
#define PROJECT_MOVIEDATABASE_H


using namespace std;

class MovieDatabase {
private:
    vector<Movie> omdb;
public:
    MovieDatabase();

    MovieDatabase(const MovieDatabase &old) {
        for (int i = 0; i < old.omdb.size(); i++) {
            this->omdb.push_back(old.omdb[i]);
        }
    }

    ~MovieDatabase() {
        //cout << "MovieDatabase destroyed" << endl;
    }

    MovieDatabase(const string &fileLocation);

    void sortFilms(int direction);

    void sortDuration();

    void sortTitleLength();

    vector<Movie> filterGenre(string genre) const;

    vector<Movie> filterAgeRating(string age) const;

    inline Movie getMovie(int index) const {
        return omdb[index];
    }

    inline void add(Movie &movie) {
        omdb.push_back(movie);
    }

    inline void add(vector<Movie> &db) {
        for (int i = 0; i < db.size(); i++) {
            this->omdb.push_back(db[i]);
        }
    }

    inline int size() const {
        return omdb.size();
    }

    friend std::ostream &operator<<(std::ostream &outputStream, const
        MovieDatabase &omdb);

};

void movieDatabaseTest();

#endif //PROJECT_MOVIEDATABASE_H
```

MovieDatabase.cpp

```
/*
By Robin Rai
V.1.0.0
Created on 05/03/2020
This file features constructors for MovieDatabase, some lambdas used for sorting,
     some filter functions, overloaded
 operators, and a test function
*/

#include <fstream>
#include <sstream>
#include "MovieDatabase.h"


using namespace std;

MovieDatabase::MovieDatabase() {};

MovieDatabase::MovieDatabase(const string &fileLocation) {

    ifstream file;
    string line;
    file.open(fileLocation);
    if (!file) {
        cout << "Unable to open file";
        exit(1); // terminate with error
    }
    while (getline(file, line)) {
        //for every good line, it makes a movie, and sets it's variables to
           whatever's on the line with the overload
        Movie temp;
        std::istringstream iss(line);
        iss >> temp;
        add(temp);
    }
    file.close();
}

void MovieDatabase::sortFilms(const int direction) {
    //cheekily uses overloaded < and >, similar to compareTo using .equals in
       java.
    //since it uses the overload, it will compare by year first, then title

    if (direction == 0 || direction == 1) {
        sort(omdb.begin(), omdb.end());
    } else {
        cout << "bad input" << endl;
    }
    if (direction == 1) {
        //When using 1/reverse order, it will invert the order of titles as well.
            So B will be before
        //A if their year is the same. I could get rid of title sorting all
           together, but I kinda like it
        reverse(omdb.begin(), omdb.end());
    }
}


void MovieDatabase::sortDuration() {
    //look! a lambda! I'm so proud. Sorts by duration, then by movie (year then
       title)
```

```cpp

    sort(omdb.begin(), omdb.end(),
         [](const Movie &mov1, const Movie &mov2) {
             if (mov1.getDuration() == mov2.getDuration()) {
                 return &mov1 < &mov2;
                 //if the duration's the same for both, compare by movie
             } else {
                 return (mov1.getDuration() < mov2.getDuration());
             }
         });
}

void MovieDatabase::sortTitleLength() {
    //look! another lambda! I'm still so proud. Sorts by title length, then by
        movie
    sort(omdb.begin(), omdb.end(),
         [](const Movie &mov1, const Movie &mov2) {
             if (mov1.getTitle().length() == mov2.getTitle().length()) {
                 return &mov1 < &mov2;
                 //if the lengths are the same, compare by movie
             } else {
                 return (mov1.getTitle().length() < mov2.getTitle().length());
             }
         });
}

vector<Movie> MovieDatabase::filterGenre(string genre) const {
    //returns a vector of only the movies with the genre specified
    vector<Movie> result;
    for (int i = 0; i < this->omdb.size(); i++) {
        if (this->omdb[i].getGenre().find(genre) != string::npos) {
            //find() will either return the position if it finds it, or npos if
                it doesn't
            //we don't care about the index at where it was found, just if it
                found it or not (npos)
            result.push_back(this->omdb[i]);
        }
    }
    return result;
}

vector<Movie> MovieDatabase::filterAgeRating(string age) const {
    //returns a vector of only the movies with the ageRating specified
    vector<Movie> result;
    for (int i = 0; i < this->omdb.size(); i++) {
        if ((age.compare(this->omdb[i].getAgeRating())) == 0) {
            //just compares strings instead of the find thing
            result.push_back(this->omdb[i]);
        }
    }
    return result;
}

std::ostream &operator<<(std::ostream &outputStream, const MovieDatabase &omdb) {
    //goes through vector of films and << them. No endl since movie's << does
        that already.
    //not inline/header file since lotsa function calls going on
    for (int i = 0; i < omdb.size(); i++) {
        outputStream << omdb.getMovie(i);
    }
    return outputStream;
}

void movieDatabaseTest() {
    //MovieDatabase badLocation("i could really do with another ice cream");
    MovieDatabase omdbTest("movieDatabaseTest.txt");

    Movie movieAdd("Ikiru", 1952, "NOT RATED", "Drama", 143, 0);

    Movie vectorAdd("Life Is Beautiful", 1997, "PG-13", "Comedy/Drama/War", 116,
       0);
    Movie vectorAdd2("Castle in the Sky", 1986, "PG", "Adventure/Animation/Family
       ", 125, 0); //bloomin' love laputa
    vector<Movie> movieVector;
    movieVector.push_back(vectorAdd);
    movieVector.push_back(vectorAdd2);

    omdbTest.add(movieAdd);
    omdbTest.add(movieVector);

    cout << "Original order:" << endl;
    cout << omdbTest << endl;

    cout << "Sorting by film (year, then title):" << endl;
    omdbTest.sortFilms(0);  //reverse sortFilms is exactly that - the same year
       will have it's title sorted backwards
    cout << omdbTest << endl;

    cout << "Sorting by duration:" << endl;
    omdbTest.sortDuration();
    cout << omdbTest << endl;

    cout << "Sorting by title length:" << endl;
    omdbTest.sortTitleLength();
    cout << omdbTest << endl;

    cout << "Filtering by genre:" << endl;
    vector<Movie> genreTest = omdbTest.filterGenre("Comedy");
    //filtering by "" will return everything, since everything contains nothing,
       and filtering a genre that doesn't
    //exist returns nothing
    MovieDatabase genreDB;
    genreDB.add(genreTest);
    cout << genreDB << endl;

    cout << "Filtering by ageRating:" << endl;
    vector<Movie> ageRatingTest = omdbTest.filterAgeRating("PG");
    //filtering by "" will return everything, since everything contains nothing,
       and filtering a ageRating that doesn't
    //exist returns nothing
    MovieDatabase ageRatingDB;
    ageRatingDB.add(ageRatingTest);
    cout << ageRatingDB << endl;


}
```

main.cpp

```cpp
/*
By Robin Rai
V.1.0.0
Created on 05/03/2020
 This file runs the program with the intended input. It runs the two test
    functions, then the program as to spec.
*/

#include <iostream>
#include "MovieDatabase.h"
#include "Movie.h"

using namespace std;

int main() {

    cout << "MOVIE TESTING: " << endl;
    movieTest();
    cout << "MOVIE DATABASE TESTING: " << endl;
    movieDatabaseTest();


    cout << "ACTUAL PROGRAM OUTPUT: " << endl;
    //initializing database

    MovieDatabase omdb("films.txt");


    //all movies in chronological order

    omdb.sortFilms(1);
    cout << endl << "All films in chronological order:" << endl;
    cout << omdb;


    //third longest film noir

    vector<Movie> genreFiltered = omdb.filterGenre("Film-Noir");

    MovieDatabase filmNoirDb;

    filmNoirDb.add(genreFiltered);

    filmNoirDb.sortDuration();

    cout << endl << "Third longest Film-Noir:" << endl;
    cout << filmNoirDb.getMovie(filmNoirDb.size() - 3) << endl;


    //eight most recent unrated

    vector<Movie> ageFiltered = omdb.filterAgeRating("UNRATED");

    MovieDatabase unratedDb;

    unratedDb.add(ageFiltered);

    unratedDb.sortFilms(0);

    cout << "Eighth most recent unrated:" << endl;
    cout << unratedDb.getMovie(unratedDb.size() - 8) << endl;
    //longest titled film

    omdb.sortTitleLength();

    cout << "Longest title: " << endl;
    cout << omdb.getMovie(omdb.size() - 1) << endl;


    return 0;
}
```