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

100238596 (pxh18ksu)

Thursday 14th May, 2020 05:56

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

```
//
// Created by james on 18/03/2020.
//

#ifndef CW2_MOVIE_H
#define CW2_MOVIE_H
#include <string>
#include <iomanip>

class Movie {

private:
    /*********  Private fields  **********/

    std::string title;
    float avg_rating{};
    int year{}, duration{};
    std::string  age_certificate;
    std::string  genre;

    struct Age_ratings // Bit field for age ratings
    {
        bool PG13 : 1, R : 1, PG : 1,
                NA : 1, NR : 1, UR : 1,
                PASS : 1, G : 1, APPR : 1,
                M : 1, X : 1, TV14 : 1;
    };
    Age_ratings age_rating = {0};


public:
    /*********  Constructors  **********/
    Movie();
    Movie(std::string title, int year, std::string age_rating, std::string genres
        ,
           int duration, float avg_rating);
    Movie(const Movie &orig);
    /*********  Destructor  **********/
    virtual ~Movie();

    //Stream based I/O using operator overloading
    friend std::istream &operator >>(std::istream &input, Movie &mov);
    friend std::ostream &operator <<(std::ostream &output, Movie &mov);

    //operators to sort Movie attributes
    friend bool operator <(Movie &mov1, Movie &mov2);
    friend bool operator >(Movie &mov1, Movie &mov2);
    friend bool operator >=(Movie &mov1, Movie &mov2);
    friend bool operator <=(Movie &mov1, Movie &mov2);

    inline std::string get_title() const { return this->title; }
    inline int get_year() const { return this->year; }
    inline std::string get_age_certificate() const { return this->age_certificate
        ; }
    inline std::string get_genres() const { return this->genre; }
    inline int get_duration() const { return this->duration; }
    inline float get_avg_rating() const { return this->avg_rating; }
    inline std::string get_age_rating() const;

    static void setTitle(std::string title) {
        title = title;
```

```
    }
61
63 };

65 inline std::string Movie::get_age_rating() const
   {
67     std::string age_rating_out;
       if (age_rating.PG13)
69         age_rating_out.assign("PG-13");
       if (age_rating.R)
71         age_rating_out.assign("R");
       if (age_rating.PG)
73         age_rating_out.assign("PG");
       if (age_rating.NA)
75         age_rating_out.assign("N/A");
       if (age_rating.NR)
77         age_rating_out.assign("NOT RATED");
       if (age_rating.UR)
79         age_rating_out.assign("UNRATED");
       if (age_rating.PASS)
81         age_rating_out.assign("PASSED");
       if (age_rating.G)
83         age_rating_out.assign("G");
       if (age_rating.APPR)
85         age_rating_out.assign("APPROVED");
       if (age_rating.M)
87         age_rating_out.assign("M");
       if (age_rating.X)
89         age_rating_out.assign("X");
       if (age_rating.TV14)
91         age_rating_out.assign("TV-14");
       return age_rating_out;
93 }

95 #endif //CW2_MOVIE_H
```

# Movie.cpp

```cpp
//
// Created by james on 18/03/2020.
//

#include "Movie.h"
#include <string>
#include <sstream>
#include <iostream>
#include <utility>


// Default constructor
Movie::Movie() = default;

Movie::Movie(const Movie &orig)
{
    this->title = orig.title;
    this->year = orig.year;
    this->age_certificate = orig.age_certificate;
    this->genre = orig.genre;
    this->duration = orig.duration;
    this->avg_rating = orig.avg_rating;
}

Movie::Movie(std::string title, int year, std::string age_certificate, std::
    string genres,
             int duration, float avg_rating) {
    this->title = title;
    // Check if year of movie is viable
    if (year < 1800 || year > 2100) {
        throw "ERROR: Couldn't create '" + title + "'. Invalid date.";
    }
    this->year = year;
    this->age_certificate = std::move(age_certificate);
    this->genre = std::move(genres);
    this->duration = duration;
    this->avg_rating = avg_rating;
}

//Destructor
Movie::~Movie(){}

//Stream based I/O using operator overloading
std::ostream &operator<<(std::ostream &output, Movie &mov) {
    return output << std::quoted(mov.get_title()) << ','
                  << mov.get_year() << ','
                  << std::quoted(mov.get_age_certificate()) << ','
                  << std::quoted(mov.get_genres()) << ','
                  << mov.get_duration() << ','
                  << std::setprecision(2) << std::fixed << mov.get_avg_rating();
}

std::istream &operator>>(std::istream &input, Movie &mov){
    int date, duration;
    float avg_rating;
    std::string title, ageRating, genres;
    char c;

    // Checks if the inputs are correct, then tries to create a movie
    if (input >> quoted(title) >> c >> date >> c >> quoted(ageRating) >>
    c >> quoted(genres) >> c >> duration >> c >> avg_rating){
```

```
61          try{

63              mov = Movie(title, date, ageRating, genres, duration, avg_rating);

65              //Switch statement in c++ uses int value, so used if statements
                    instead

67              if (ageRating == "PG-13")
                {
69                  mov.age_rating.PG13 = true;
                }
71              if (ageRating == "R")
                {
73                  mov.age_rating.R = true;
                }
75              if (ageRating == "PG")
                {
77                  mov.age_rating.PG = true;
                }
79              if (ageRating == "N/A")
                {
81                  mov.age_rating.NA = true;
                }
83              if (ageRating == "NOT RATED")
                {
85                  mov.age_rating.NR = true;
                }
87              if (ageRating == "UNRATED")
                {
89                  mov.age_rating.UR = true;
                }
91              if (ageRating == "PASSED")
                {
93                  mov.age_rating.PASS = true;
                }
95              if (ageRating == "G")
                {
97                  mov.age_rating.G = true;
                }
99              if (ageRating == "APPROVED")
                {
101                 mov.age_rating.APPR = true;
                }
103             if (ageRating == "M")
                {
105                 mov.age_rating.M = true;
                }
107             if (ageRating == "X")
                {
109                 mov.age_rating.X = true;
                }
111             if (ageRating == "TV-14")
                {
113                 mov.age_rating.TV14 = true;
                }
115
            } catch (const std::exception &exc) {
117             std::cerr << "Error: Failed to create " + title + exc.what() << std::
                    endl;
            }
119     } else {
            input.clear(std::ios_base::failbit);
121     }
```

```
123         return input;
    }
```

# MovieDatabase.h

```
//
// Created by james on 08/04/2020.
//

#include <set>
#include <vector>
#include <functional>
#include "Movie.h"

//#ifndef CW2_MOVIEDATABASE_H
//#define CW2_MOVIEDATABASE_H

class MovieDatabase{
private:
    std::vector<Movie> mdb;

public:
    /*********  Constructors  **********/
    MovieDatabase();
    MovieDatabase(std::string file);
    MovieDatabase(const MovieDatabase &orig);
    MovieDatabase(std::vector<Movie> movie_list);

    /*********  Queries  **********/
    bool addMovie(Movie movie);
    void delete_movie(const Movie& movie);

    /*********  Destructor  **********/
    virtual ~MovieDatabase();

    inline std::vector<Movie> getDB(){
        return this->mdb;
    }

    template <typename Compare>
    void sort(const Compare &comp)
    {
        if (!mdb.empty()){
            std::sort(mdb.begin(), mdb.end(), comp);
        } else {
            std::cerr << "Movie Database is empty - please populate and then sort
                " << std::endl;
        }
    }

    //Stream based I/O using operator overloading
    friend std::ostream &operator<<(std::ostream &output, MovieDatabase &db);
    friend std::istream &operator>>(std::istream &input, MovieDatabase &db);

};
```

# MovieDatabase.cpp

```
//
// Created by james on 08/04/2020.
//

#include <algorithm>
#include <sstream>
#include <iostream>
#include <ostream>
#include <iterator>
#include <string>
#include <regex>
#include <fstream>
#include <tgmath.h>
#include "MovieDatabase.h"

MovieDatabase::MovieDatabase() {}

MovieDatabase::MovieDatabase(const MovieDatabase &orig)
{
    mdb = orig.mdb;
}

MovieDatabase::~MovieDatabase()
{
    mdb.clear();
}

bool MovieDatabase::addMovie(Movie mov){
    try{
        this->mdb.push_back(mov);
        return true;
    }catch (const std::exception &exc){
        std::cout << "An error occurred adding the movie." << std::endl;
        return false;
    }
}

void MovieDatabase::delete_movie(const Movie& movie) {
    if (mdb.empty()) return;
}

//Stream based I/O using operator overloading
std::istream &operator>>(std::istream &input, MovieDatabase &db) {
    std::string line;
    //std::getline(input, line);
    line.append("\n");
    //capture each line of the txt file
    while (getline(input, line))
    {

        Movie movie;

        std::istringstream iss(line);

        iss >> movie;
        //add movie to the database
        db.addMovie(movie);
    }
    std::cout << "Movies successfully read in \n";
    return input;
}
```

```
62
std::ostream &operator<<(std::ostream &output, MovieDatabase &db){
64      for(Movie mov : db.mdb){
            output << mov << std::endl;
66      }
        return output;
68  }
```

## main.cpp

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <sstream>
#include "MovieDatabase.h"

int main() {

    std::cout << "Task 1: Read in movies, create objects of them, and store them
       in a MovieDatabase \n";
    MovieDatabase *movieDatabase = new MovieDatabase();
    std::ifstream input_file("films.txt");

    if (input_file.is_open()) { input_file >> *movieDatabase; }

    //Test Default constructor
    //Movie LoTR("Lord of the Rings", 2005, "PG-13", "Action/Adventure/Fantasy
       ",140, 6);
    //movieDatabase->addMovie(LoTR);
    //std::cout << *movieDatabase;

    std::cout << "Task 2: Sort movies in chronological order and display them \n"
       ;

    try
    {
        //Lambda sort
        auto sortAscending = [] (Movie const &m1, Movie const &m2) -> bool
                {
                    return (m1.get_year() < m2.get_year());
                };
        movieDatabase->sort(sortAscending);

        std::cout << *movieDatabase;
    }
    catch (std::exception& e)
    {
        std::cout << e.what() << '\n';
    }

    std::cout << "Task 3: Display the third longest Film-Noir \n";
    // Get only the movies which are film-noir and then sort them based on
       duration
    // Waste of resources sorting the entire vector

    try
    {
        MovieDatabase *movieDatabase2 = new MovieDatabase();
        for (auto & movie : movieDatabase->getDB()) {
            if(movie.get_genres().find("Film-Noir") != std::string::npos){
                movieDatabase2->addMovie(movie);
            }
        }

        //Lambda sort
        auto sortDuration = [] (Movie const &m1, Movie const &m2) -> bool
        {
            return (m1.get_duration() < m2.get_duration());
        };
        movieDatabase2->sort(sortDuration);
```

```
58          std::cout << (movieDatabase2->getDB())[2] << "\n";

60      }
        catch (std::exception& e)
62      {
            std::cout << e.what() << '\n';
64      }

66      std::cout << "Task 4: Display the eighth most recent UNRATED film \n";

68      try
        {
70          MovieDatabase *movieDatabase3 = new MovieDatabase();
            for (auto & movie : movieDatabase->getDB()) {
72              if(movie.get_age_certificate().find("UNRATED") != std::string::npos){
                    movieDatabase3->addMovie(movie);
74              }
            }
76
            //Lambda sort
78          auto sortAscending = [] (Movie const &m1, Movie const &m2) -> bool
            {
80              return (m1.get_year() < m2.get_year());
            };
82          movieDatabase3->sort(sortAscending);

84          std::cout << (movieDatabase3->getDB())[movieDatabase3->getDB().size() -
                8] << "\n";

86      }
        catch (std::exception& e)
88      {
            std::cout << e.what() << '\n';
90      }

92      std::cout << "Task 5: Display the film with the longest title \n";

94      try
        {
96          //Lambda sort
            auto sortAscendingTitle = [] (Movie const &m1, Movie const &m2) -> bool
98          {
                return (m1.get_title().length() > m2.get_title().length());
100         };
            movieDatabase->sort(sortAscendingTitle);
102
            std::cout << (movieDatabase->getDB())[0] << "\n";
104
        }
106     catch (std::exception& e)
        {
108         std::cout << e.what() << '\n';
        }
110


112     return 0;
    }
```