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

100263247 (uxk18qau)

Tuesday 28th April, 2020 14:50

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 danny on 03/04/2020.
//

#include <vector>
#include <string>
#include <iostream>
#include <map>

#pragma once
using namespace std;

class Movie{
public:
    enum certificateEnum {//enum of all the certificates that are in the films.txt file
        PG_13,
        PG,
        APPROVED,
        R,
        NOT_RATED,
        G,
        UNRATED,
        PASSED,
        NA,
        TV14,
        M,
        X
    };

    static const map<string, certificateEnum> certificateStringToEnum;//map for converting the string to the enum type
    static void tester();//test harness

    Movie(const string& title, int releaseYear, const string& certificate, const string& genres, int duration, int averageRating);//constructs from all the attributes
    explicit Movie(const string& line);//creates a movie from the line in the file
    Movie();//empty movie constructor
    /**
     * Gets the title of the movie
     * @return the movie title
     */
    string getTitle() const; //gets the title of the movie
    /**
     * Gets the year of release of the film
     * @return the release year
     */
    int getReleaseYear() const;//gets the release year of the movie
    /**
     * Gets the certificate value of the movie
     * @return certificate
     */
    certificateEnum getCertificate() const;//gets the certificate of the movie
    /**
     * Gets all genres as a string seperated by '/'
     * @return genre string
     */
    string getGenres() const;//gets the genre string of the movie
    /**
```

```
57          * Gets the duration of the movie in minutes
            * @return duration of the movie
59          */
        int getDuration() const;//gets the duration of the movie
61      /**
            * Gets the average rating of the movie
63          * @return average rating
            */
65      int getAverageRating() const;//gets the average rating of the movie
        /**
67          * Finds if a movie is of a specified genre
            * Using find function
69          * @param genreToMatch desired genre
            * @return true if the movie is of the desired genre
71          */
        bool hasGenre(const string& genreToMatch) const;//gets if the movie is of the
            specified genre
73
        friend ostream& operator<< (ostream& out, const Movie& movie);//override the
            operator
75      bool operator< (const Movie& other) const {//overrides the less than operator
            return this->getReleaseYear() < other.getReleaseYear();//oldest to most
                recent
77      }
        struct CompareMoviesByDuration { //struct used to compare movies by duration
            using a functor
79          bool operator () (const Movie& movie1, const Movie& movie2) const {
                return movie1.getDuration() > movie2.getDuration();//longest to
                    shortest
81          }
        };
83  private:
        string m_title; //title of the movie
85      int m_releaseYear;//year of release of the movie
        certificateEnum m_certificate;//certificate of who can watch the movie
87      string m_genres;//genres the movie fits into
        int m_duration;//duration of the film in minutes
89      int m_averageRating;//average viewer rating
        static vector<string> splitString(const string& str, const string& seperator)
            ; //split string function
91      static string searchForKey(const certificateEnum& certificateEnum);
    };
```

Movie.cpp

```
//
// Created by danny on 03/04/2020.
//

#include "Movie.h"
using namespace std;

const map<string, Movie::certificateEnum> Movie::certificateStringToEnum = {
        {"PG-13", PG_13},
        {"PG", PG},
        {"APPROVED", APPROVED},
        {"R", R},
        {"NOT RATED", NOT_RATED},
        {"G", G},
        {"UNRATED", UNRATED},
        {"PASSED",PASSED},
        {"N/A",NA},
        {"TV-14",TV14},
        {"M",M},
        {"X",X}
};

/**
 * Test harness
 * Constructs 2 movies, a and b using two different constructors
 * Prints the 2 movies
 * See if each movie has genre action
 * See if each movie has the genre fantasy
 */
void Movie::tester(){
    Movie a = Movie("Indiana Jones and the Last Crusade",1989,"PG-13","Action/
        Adventure/Fantasy",127,0);
    Movie b = Movie(R"("Aliens",1986,"R","Action/Adventure/Sci-Fi",137,0)");
    cout << "a: ";
    cout << a;
    cout << "b: ";
    cout << b;
    cout << "a has genre \"Action\": " << a.hasGenre("Action") << "\n";
    cout << "b has genre \"Action\": " << b.hasGenre("Action")<< "\n";
    cout << "a has genre \"Fantasy\": " << a.hasGenre("Fantasy")<< "\n";
    cout << "b has genre \"Fantasy\": " << b.hasGenre("Fantasy")<< "\n";
}

/**
 * Constructor
 * Constructs a movie from all the attributes passed in
 * @param title, title of the movie
 * @param releaseYear, year or release of the movie
 * @param certificate, certificate of the movie
 * @param genres, genres of the movie
 * @param duration, duration of the movie
 * @param averageRating, average rating of the movie
 */
Movie::Movie(const string& title, int releaseYear, const string& certificate,
    const string& genres, int duration, int averageRating) {//Constructor
    m_title = title;
    m_releaseYear = releaseYear;
    m_certificate = Movie::certificateStringToEnum.at(certificate);
    m_genres = genres;
    m_duration = duration;
    m_averageRating = averageRating;
```

```
60 }

62 /**
    * Constructor
64  * Creates a movie from a line in a file
    * Has to be in the correct format
66  * @param line the line of the file
    */
68 Movie::Movie(const string& line){
        vector<string> lineSplit = splitString(line,"\"");//splits the line by "
70      m_title = lineSplit[1];//splitting makes the title the 1st element
        m_releaseYear = stoi(lineSplit[2].substr(1,lineSplit[2].size() - 2));//remove
            the commas and convert to integer
72      m_certificate = Movie::certificateStringToEnum.at(lineSplit[3]);//splitting
            makes the certificate 3rd
        m_genres = lineSplit[5];//all the genres are stored together and a hasGenre
            function was created
74      vector<string> runtimeAndRatingSplit = splitString(lineSplit[6],",");//splits
            the already split line by commas to seperate the runtime and rating
        m_duration = stoi(runtimeAndRatingSplit[1]);//converts to integer
76      m_averageRating = stoi(runtimeAndRatingSplit[2]);//converts to integer
    }
78
    Movie::Movie(){}
80
    //accessor methods
82  string Movie::getTitle() const{
        return m_title;
84  }

86  int Movie::getReleaseYear() const{
        return m_releaseYear;
88  }

90  Movie::certificateEnum Movie::getCertificate() const{
        return m_certificate;
92  }

94  string Movie::getGenres() const{
        return m_genres;
96  }

98  int Movie::getDuration() const{
        return m_duration;
100 }

102 int Movie::getAverageRating() const{
        return m_averageRating;
104 }

106 bool Movie::hasGenre(const string& genreToMatch) const{
        size_t found = getGenres().find(genreToMatch);//gets the position of the
            start of the the string to find
108     return found != string::npos; // Return true if the genreToMatch was found
    }
110
    /**
112  * Prints out the movie overloading the << operator
     * @param out output stream
114  * @param movie movie
     * @return the output stream full with the movie
116  */
    ostream& operator<< (ostream &out, const Movie &movie) {
```

```
118     out << "\"" << movie.getTitle() << "\"," << to_string(movie.getReleaseYear())
            << ",\"" << Movie::searchForKey(movie.getCertificate()) << "\",\"" <<
        movie.getGenres() << "\"," << to_string(movie.getDuration()) << "," <<
        to_string(movie.getAverageRating()) << "\n";
    return out; // return ostream so I can chain calls to operator<< used for the
120 }

122 /**
     * Splits the string to split on each separator string into a vector
124  * @param str string to split
     * @param seperator string to split on
126  * @return vector of all the substrings split
     */
128 vector<string> Movie::splitString(const string& str, const string& seperator) {
    vector<string> parts;//parts of the string after split
130     size_t pos = 0;//start at the beginning
    while (true) {
132         size_t newPos = str.find(seperator, pos);//try to find the place of the
            next separator
        if (newPos == string::npos) {// No more separators
134             parts.push_back(str.substr(pos));//push back last substring
            break;//exit the while
136         }
        // Found separator at newPos
138         parts.push_back(str.substr(pos, newPos - pos));//push back new substring
        pos = newPos + seperator.length();//jumps to the end of the separator
140     }
    return parts;
142 }

144 /**
     * Search for the key string of the certificate from the certificate enum
146  * @param certificateEnum certificate enum to match
     * @return the string, matching the certificate enum
148  */
string Movie::searchForKey(const Movie::certificateEnum& certificateEnum){
150     string key;
    for (auto &item : Movie::certificateStringToEnum) {//goes through the map
152         if (item.second == certificateEnum) {//compares the second item (the
            value) to
             key = item.first; //the key is set
154             break; //To stop searching
        }
156     }
    return key;
158 }
```

MovieDatabase.h

```
//
// Created by danny on 03/04/2020.
//

#include <string>
#include <vector>
#include <fstream>

#include "Movie.h"
#pragma once
using namespace std;

class MovieDatabase{
public:
    static void tester();//test harness
    explicit MovieDatabase(const string& fileName);//constructor from file name
    MovieDatabase();//constructor of empty database
    void add(const Movie& movie);//add a movie
    void resize(const size_t& newSize);
    Movie get(int index) ;//get a movie in position index
    int size() const;//the size of the database

    friend ostream& operator<< (std::ostream &out, const MovieDatabase &md);

    void sortByTitleLength();//sorts the database by title length
    void sortByReleaseYear();//sorts the movies by release year
    MovieDatabase filterByCertificate(const string& certificateToMatch);//gives a
        new database of movies with a specific certificate
    MovieDatabase filterByGenre(const string& genreToMatch);//gives a new
       database with all the movies that have a specific genre
    void sortByDuration();//sorts the database by duration
    void sortByAverageRating();//sorts the database by average rating
private:
    vector<Movie> m_db;//vector of movies in the database
};
```

MovieDatabase.cpp

```
1  //
   // Created by danny on 03/04/2020.
3  //

5  #include <algorithm>
   #include "MovieDatabase.h"
7
   /**
9   * Test harness
    *
11  * Constructs a database from the file
    * Sorts the database by release year
13  * Displays the size of the database
    * Filters by PG certificate
15  * Displays the size of the database to prove the difference
    * Sorts by the title length of the movies
17  * Filters by the genre of comedy
    * Displays the size to show the difference in size
19  * Sorts by average rating
    * Sorts by duration
21  * Prints the longest film
    */
23 void MovieDatabase::tester(){
       MovieDatabase database = MovieDatabase("films.txt");
25     cout << "In from file\n";
       cout << database;
27     database.sortByReleaseYear();
       cout << "Sorted by release year\n";
29     cout << database;
       cout << "Size: " << database.size() << "\n";
31     database = database.filterByCertificate("PG");
       cout << "Filter by Certificate \"PG\"\n";
33     cout << "Size: " << database.size() << "\n";
       cout << database;
35     database.sortByTitleLength();
       cout << "Sort by title length\n";
37     cout << database;
       database = database.filterByGenre("Comedy");
39     cout << "filter by genre \"Comedy\"\n";
       cout << "Size: " << database.size() << "\n";
41     cout << database;
       database.sortByAverageRating();
43     cout << "Sort by average rating\n";
       cout << database;
45     database.sortByDuration();
       cout << "Sort by duration\n";
47     cout << database;
       cout << "First (longest duration): " << database.get(0) << "\n";
49 }

51 /**
    * Constructor
53  * Reads in the file line by line if the file exists
    * While there is a line, it constructs a movie object and adds it to the
        database
55  * Closes the file
    * @param fileName, the name of the file to build the database from
57  */
   MovieDatabase::MovieDatabase(const string& fileName){
59     ifstream file(fileName);
       if (file.is_open()){
```

```
61          string line;
            while (getline(file, line)) {//while there is still another line
63              m_db.emplace_back(line);  // construct a new Movie directly into the
                    database
            }
65          file.close();
        } else {
67          cout << "Error: Unable to find file " << fileName << endl;
        }
69  }

71  /**
     * Constructor
73   * Creates a new movie database obj by creating a brand new empty vector
     */
75  MovieDatabase::MovieDatabase():m_db(){
    }
77
    /**
79   * Modifier
     * Adds a movie to the database
81   * @param movie to add
     */
83  void MovieDatabase::add(const Movie& movie) {
        m_db.push_back(movie); // Add to the end of the database
85  }

87  /**
     * Modifier
89   * Resizes the database vector to new size
     * @param newSize, the new size to set the vector to
91   */
    void MovieDatabase::resize(const size_t& newSize){
93      m_db.resize(newSize);//resize the new vector
    }
95
    /**
97   * Accessor
     * Gets the Movie at index in the vector
99   * @param index to get
     * @return Movie at index
101  */
    Movie MovieDatabase::get(int index){
103     return m_db.at(index);//a vector has this method at already however the
            vector is private so needs its own accessor method
    }
105
    /**
107  * @return The number of elements in the vector
     */
109 int MovieDatabase::size() const{
        return m_db.size();
111 }

113 /**
     * Prints out the movie database overloading the << operator
115  * @param out output stream
     * @param movieDatabase movie database
117  * @return the output stream full with the movie database
     */
119 ostream& operator<< (std::ostream &out, const MovieDatabase &movieDatabase) {
        for (auto const& movie : movieDatabase.m_db) {
121         out << movie;
```

```
    }
123     return out; // return std::ostream so we can chain calls to operator<<
}
125
/**
127  * I implemented 4 different methods to sort the database on the 4 different
       fields
     * For the title length I used a function
129  * For the release year I overloaded the < operator in the movie header file
     * For the duration I used a functor
131  * For the average rating I used a lambda
     * I wanted to explore different ways of using the std::sort method
133  * And the 2 different methods to filter the database
     * For Filtering by genre I used for loop with an if statement
135  * For filtering by certificate I used a copy_if function with iterator
     */
137 namespace {
    /**
139      * Comparator function for movie title length
         * The function that returns movie1 boolean that I am using in
            SortMoviesByTitleLength
141      * @param movie1 referance to first movie
         * @param movie2 reverance to second movie
143      * @return if the first title is longer than the second
         */
145     bool CompareMoviesByTitleLength(const Movie& movie1, const Movie& movie2) {
            return (movie1.getTitle().length() > movie2.getTitle().length());
147     }
}
149
/**
151  * The actual sorting function that would be called to sort the database obj
     * This uses the comparator function above as a third parameter in std::sort
153  * highest to lowest
     */
155 void MovieDatabase::sortByTitleLength(){//function
    sort(m_db.begin(),m_db.end(),CompareMoviesByTitleLength);
157 }

159 /**
     * Sorting function to sort the movies by year of release
161  * Uses a relational comparator override therefore only two parameters in std::
       sort
     * lowest to highest
163  */
void MovieDatabase::sortByReleaseYear(){
165     sort(m_db.begin(),m_db.end());
}
167
/**
169  * Sorts the database by duration of the movies
     * Create a functor (i.e. a callable object) that implements operator()
171  * Such that when it is called it returns true if it's  first parameter
     * should come before its second parameter
173  * Uses a functor
     */
175 void MovieDatabase::sortByDuration(){
    Movie::CompareMoviesByDuration compareFunctor; //Create a functor that
       implements operator()
177     sort(m_db.begin(),m_db.end(), compareFunctor);// Call sort, passing the
       functor object as the third parameter
}
179
```

```
/**
 * Unused for the task on blackboard
 * Sorts the database by average rating
 * This uses a lambda to sort
 * highest to lowest
 */
void MovieDatabase::sortByAverageRating() {
    sort(m_db.begin(), m_db.end(),[](const Movie & movie1, const Movie & movie2)
        -> bool {//use an lamda to sort
         return movie1.getAverageRating() > movie2.getAverageRating();
    });
}

/**
 * Creates a new movie database of the current size
 * Directly edits the new databases vector of movies and copies the movies in
 * Resizes the new databases vector of movies
 * @param certificateToMatch the certificate that all the movies must have
 * @return a movie database of all the movies that have this certificate
 */
MovieDatabase MovieDatabase::filterByCertificate(const string& certificateToMatch
    ){
    MovieDatabase newDatabase = MovieDatabase();
    newDatabase.resize(m_db.size());
    auto it = copy_if(m_db.begin(), m_db.end(), newDatabase.m_db.begin(), [&
        certificateToMatch](const Movie& movie) {//use copy if and a lambda
         return (movie.getCertificate() == Movie::certificateStringToEnum.at(
            certificateToMatch));//filter by the certificate
    });
    newDatabase.resize(distance(newDatabase.m_db.begin(),it));//resizes the array
         in the database to the correct size so no empty movies
    return newDatabase;
}

/**
 * Constructs a new movie database obj
 * For each movie in the current database obj
 * If the movie has that genre in
 * Add it to the new database
 * @param genreToMatch the genre to match
 * @return a new database obj of all the films in the desired genre
 */
MovieDatabase MovieDatabase::filterByGenre(const string& genreToMatch){
    MovieDatabase newdb = MovieDatabase();//make a empty new database
    for(auto movie: m_db){
        if (movie.hasGenre(genreToMatch)){//if the movie contains the desired
            genre
             newdb.add(movie);//add the movie to the database
        }
    }
    return newdb;
}
```

main.cpp

```cpp
#include "Movie.h"
#include "MovieDatabase.h"

int main() {
    //Movie::tester();//movie class test harness
    //MovieDatabase::tester();//movie database class test harness

    // 1. Read in the database from the file films.txt, using the relative path âĂŸfilms.txtâĂİ,
    // provided via BlackBoard (when using CLion, the program will expect to find the file
    // in the cmake-build-debug directory). This is necessary to ensure that the program
    //runs correctly using PASS.
    cout << "Task 1\nReading in films\n" << endl;
    MovieDatabase database = MovieDatabase("films.txt");//creates a movie database from file

    // 2. Display the entire collection of movies, arranged in chronological order. The movies
    // must be displayed in the same format in which they appear in films.txt.
    cout << "Task 2\nSort by the release year" << endl;
    database.sortByReleaseYear();//sorts the database lowest to highest of year of release
    cout << database << endl;//prints the database in order

    // 3. Display the third longest Film-Noir
    cout << "Task 3\nThird longest Film-Noir" << endl;
    MovieDatabase filmNoir = database.filterByGenre("Film-Noir");//creates a new database with all the Film-Noir films in
    filmNoir.sortByDuration();//sorts the filmnoir database by duration longest to shortest
    cout << filmNoir.get(2) << endl;//counting from 0 so for the task I need to get the second element and print it

    //4. Display the eighth most recent UNRATED film
    cout << "Task 4\nEighth most recent UNRATED film" << endl;
    MovieDatabase unratedFilms = database.filterByCertificate("UNRATED");//gets all the unrated films in a new database
    cout << unratedFilms.get(unratedFilms.size()-8) << endl;//already sorted by release year, but needs the size of it subtract 8 to get most recent

    //5. Display the film with the longest title.
    cout << "Task 5\nLongest title" << endl;
    database.sortByTitleLength();//sort the whole database by title length
    cout << database.get(0);//prints the longest title length movie

    return 0;
}
```