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

100214063 (`uyx17kku`)

Sunday 2nd August, 2020 13:57

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

```
1
2
3  #ifndef MOVIE_H
   #define MOVIE_H
5
   #include <cstdlib>
7  #include <string>
   #include <fstream>
9  #include <iostream>
   #include <iomanip>
11 #include <algorithm>
   #include <vector>
13 #include <sstream>

15 using namespace std;

17 #include "MovieLink.h"

19 class Movie {
   private:
21     string title;
       unsigned int year;
23     MovieCertificate *cert;
       vector<MovieGenre *> genre;
25     unsigned int duration;
       MovieDatabase *parent;
27     unsigned int reviewCount;
       unsigned int reviewPoints;
29 public:
       friend class MovieDatabase;
31
       friend class Movie;
33
       // constructors / deconstructors
35
       Movie();
37
       Movie(Movie *cloneMovie, MovieDatabase *newParent);
39
       Movie(MovieDatabase *newParent);
41
       // setters
43     void setCert(MovieCertificate *newCert);

45     void setCert(string newCertName);

47     void setTitle(string newTitle);

49     void setYear(unsigned int newYear);

51     void setDuration(unsigned int newDuration);

53     void addGenre(MovieGenre *newGenre);

55     void addGenre(string newGenreName);

57     void removeGenre(MovieGenre *oldGenre);

59     void setReviewCount(unsigned int newReviewCount); //for future reviews
       void setReviewPoints(unsigned int newReviewPoints);
61
```

```
    void setReviewAverage(double newReviewAverage);

    void addScore(int score);


    // getters
    MovieDatabase *getParent() const;

    string getCert() const;

    string getTitle() const;

    unsigned int getYear() const;

    unsigned int getDuration() const;

    //check for genre and cert
    bool hasGenre(MovieGenre *findGenre);

    bool hasCert(MovieCertificate *findCert);

    //returns a string of the multiple genres
    string getGenresStr() const;

    //number of genres
    int getGenreCount() const;

    //when reviews are added to the database
    unsigned int getReviewCount() const;

    double getAverageScore() const;

};


//Stream based I/O using operator overloading
inline std::ostream &operator<<(ostream &os, Movie &m) {
    return os << '"' << m.getTitle() << '"' << ',' << m.getYear() << ',' <<
              '"' << m.getCert() << '"' << ',' << '"' << m.getGenresStr() <<
              '"' << ',' << m.getDuration() << ',' << m.getAverageScore();
};


inline std::stringstream &operator>>(stringstream &s1, Movie *m1) {
    string discard;
    string Title, Year, Cert, Genres, Duration, AverageReview;
    // getting the corresponding values for their variable
    getline(s1, discard, '"');
    getline(s1, Title, '"');
    getline(s1, discard, ',');
    getline(s1, Year, ',');
    getline(s1, discard, '"'); //due to "" in stream
    getline(s1, Cert, '"');
    getline(s1, discard, '"');
    getline(s1, Genres, '"');
    getline(s1, discard, ',');
    getline(s1, Duration, ',');
    getline(s1, AverageReview, ',');

    // setting
    m1->setTitle(Title);
    m1->setYear((unsigned int) stoi(Year));
    m1->setCert(Cert);
```

```
125     m1->setDuration((unsigned int) stoi(Duration));
        m1->setReviewAverage(stoi(AverageReview));
127     m1->setCert(Cert);


129
        // add the genres
131     stringstream ssGenres(Genres);
        string Genre;
133
        while (ssGenres) {
135         getline(ssGenres, Genre, '/');
            m1->addGenre(Genre);
137     }
        return s1;
139 };


141
    //inspired from https://stackoverflow.com/questions/37608526/not-declared-in-
        scope-friend-comparator-class-for-priority-queue-c
143 //Handling the comparison of the films using methods in database
    inline bool operator<(const Movie &a, const Movie &b) {
145     return CompareFilm::movieLT(a, b, *a.getParent());
    }
147
    inline bool operator>(const Movie &a, const Movie &b) {
149     return CompareFilm::movieGT(a, b, *a.getParent());
    }
151
    inline bool operator==(const Movie &a, const Movie &b) {
153     return CompareFilm::movieEQ(a, b, *a.getParent());
    }
155
    inline bool operator!=(const Movie &a, const Movie &b) {
157     return !CompareFilm::movieEQ(a, b, *a.getParent());
    }
159
    inline bool operator<=(const Movie &a, const Movie &b) {
161     return !CompareFilm::movieGT(a, b, *a.getParent());
    }
163
    inline bool operator>=(const Movie &a, const Movie &b) {
165     return !CompareFilm::movieLT(a, b, *a.getParent());
    }
167


169 #endif /* MOVIE_H */
```

Movie.cpp

```
1
3  #include "Movie.h"
5
   //Construct for copying movie to another database to get a new list for certain
       parameters
7  Movie::Movie(Movie *cloneMovie, MovieDatabase *newParent) {
       this->parent = newParent;
9      this->setTitle(cloneMovie->getTitle());
       this->setCert(cloneMovie->getCert());
11     this->setYear(cloneMovie->getYear());
       this->setDuration(cloneMovie->getDuration());
13     this->setReviewAverage(cloneMovie->getAverageScore());
       //as there is multiple genres in some cases
15     int i = 0;
       int a = cloneMovie->getGenreCount();
17     while (++i < a) {
           this->addGenre(cloneMovie->genre[i]);
19     }
   };
21
   //movie created with reference to the parent database
23 Movie::Movie(MovieDatabase *newParent) {
       this->parent = newParent;
25 };

27 //Getters
   //return pointer to parent db
29 MovieDatabase *Movie::getParent() const {
       return parent;
31 };

33 string Movie::getTitle() const {
       return this->title;
35 };

37 string Movie::getCert() const {
       return this->cert->toString();
39 };

41 unsigned int Movie::getYear() const {
       return this->year;
43 };

45 unsigned int Movie::getDuration() const {
       return this->duration;
47 };

49 //returns the multiple genres as string
   string Movie::getGenresStr() const {
51     int x = this->genre.size();
       if (x == 0) {
53         return "";
       };
55     stringstream s1;
       s1 << this->genre[0]->toString();
57     if (x > 1) {
           int i = 0;
59         while (++i < x) {
               s1 << '/' << this->genre[i]->toString();
```

```
61          };
        };
63      return s1.str();
    };
65
    //get number of genres a film has
67  int Movie::getGenreCount() const {
        return genre.size();
69  };

71  //not used but for when scores are added to films
    double Movie::getAverageScore() const {
73      if (this->reviewCount == 0) {
            return 0;
75      }
        return ((double) this->reviewPoints) / (this->reviewCount);
77  };

79  //setters
    void Movie::setTitle(string newTitle) {
81      this->title = newTitle;
    };
83
    void Movie::setYear(unsigned int newYear) {
85      this->year = newYear;
    };
87
    void Movie::setCert(MovieCertificate *newCert) {
89      this->cert = newCert;
    };
91
    void Movie::setCert(string newCertName) {
93      this->setCert(MovieDatabaseLink::getCert(newCertName, *(this->getParent())));

95  };

97  void Movie::setDuration(unsigned int newDuration) {
        this->duration = newDuration;
99  };

101 void Movie::setReviewAverage(double newReviewAverage) {
        this->reviewPoints = (unsigned int) newReviewAverage;
103 };


105
    //add genre to  genre vector
107 void Movie::addGenre(MovieGenre *newGenre) {
        if (!hasGenre(newGenre)) {
109         (this->genre).push_back(newGenre); //add genre to end of vector
        }
111 };

113 //checks the parent database if its a new genre and added if it is
    void Movie::addGenre(string newGenreName) {
115     this->addGenre(MovieDatabaseLink::getGenre(newGenreName, *(this->getParent())
            ));
    };
117
    //checks for genre and cert
119 bool Movie::hasGenre(MovieGenre *findGenre) {
        int i = -1;
121     int ii = genre.size();
        while (++i < ii) {
```

```
123             if (genre[i] == findGenre) {
                    return true;
125             }
        }
127     return false;
    }
129
    bool Movie::hasCert(MovieCertificate *findCert) {
131     if (cert == findCert) {
            return true;
133     }
        return false;
135 }
```

MovieDatabase.h

```
2
#ifndef MOVIEDATABASE_H
4 #define MOVIEDATABASE_H

6 #include <cstdlib>
#include <string>
8 #include <fstream>
#include <iostream>
10 #include <iomanip>
#include <algorithm>
12 #include <vector>
#include <sstream>
14
using namespace std;
16

18 #include "Movie.h"

20

22 class MovieDatabase {
private:
24     std::vector<Movie> filmDB;
public:
26     enum SortFields {
           title, year, certificate, genre, duration, average_rating, title_length
28     };
       std::vector<string> titles;
30     std::vector<Movie *> movies;
       std::vector<MovieCertificate *> certificates;
32     std::vector<MovieGenre *> genres;
       SortFields sortField;
34     bool sortDesc;

36 public:
       friend class MovieDatabase;
38
       //queries
40     bool addMovie(Movie *newMovie);

42     bool addMovie(stringstream &CSVlinestream);

44     Movie *getIndex(int i) const;

46     SortFields getSortField() const;

48     void sortDatabase(SortFields field, bool desc);

50     MovieCertificate *getCert(string certStr);

52     MovieGenre *getGenre(string genreStr);

54     MovieDatabase *genreList(MovieGenre *findGenre);

56     MovieDatabase *certList(MovieCertificate *findCert);

58     long count() const;

60     string movieString(Movie &mp) const;
```

```
62      //constructor
        MovieDatabase();
64  };

66  inline std::ifstream &operator>>(ifstream &is, MovieDatabase &filmDB) {
        string lineString;
68      while (getline(is, lineString)) {
            stringstream lineStream(lineString);
70          filmDB.addMovie(lineStream);
        }
72      cout << filmDB.count() << " films got added to database\n";
        return is;
74  }

76  //reading and printing moviestring
    inline std::ofstream &operator<<(std::ofstream &os, const MovieDatabase &filmDB)
       {
78      int a = filmDB.count();
        if (a < 1) {
80          return os;
        }
82
        os << filmDB.movieString(*filmDB.getIndex(0));
84      int i = 0;
        while (++i < a) {
86          os << "\r\n" << filmDB.movieString(*filmDB.getIndex(i));

88      }
        return os;
90  }

92  inline std::ostream &operator<<(std::ostream &os, const MovieDatabase &filmDB) {
        int b = filmDB.count();
94      if (b < 1) {
            return os;
96      }
        os << filmDB.movieString(*filmDB.getIndex(0));
98      int i = 0;
        while (++i < b) {
100         os << "\r\n" << filmDB.movieString(*filmDB.getIndex(i));
        }
102     return os;
    }
104
    #endif /* MOVIEDATABASE_H */
```

MovieDatabase.cpp

```
/*

 */
#include "MovieDatabase.h"

//Constructor for db
MovieDatabase::MovieDatabase() {
    sortDesc = false;
    sortField = title; // automatically set for alphabetical
}

//return enum for sortfield
MovieDatabase::SortFields MovieDatabase::getSortField() const {
    return MovieDatabase::sortField;
};


//creating movie database for genre+ certificate
MovieDatabase *MovieDatabase::genreList(MovieGenre *findGenre) {
    MovieDatabase *gDB = new MovieDatabase();
    gDB->genres = this->genres;
    int i = 0;
    int a = movies.size();
    Movie *r = movies[0];
    while (++i < a) {
        if (movies[i]->hasGenre(findGenre)) {
            /*
             * Movies have to be cloned, as they need to point to the
             * new Movie Database to get sort settings
             */
            Movie *clone = new Movie(movies[i], gDB);
            gDB->addMovie(clone);
        }
    }
    return gDB;
};

MovieDatabase *MovieDatabase::certList(MovieCertificate *findCert) {
    MovieDatabase *cDB = new MovieDatabase();
    cDB->certificates = this->certificates;
    int i = 0;
    int b = movies.size();
    Movie *r = movies[0];
    while (++i < b) {
        if (movies[i]->hasCert(findCert)) {
            /*
             * Movies have to be cloned, as they need to point to the
             * new Movie Database to get sort settings
             */
            Movie *clone = new Movie(movies[i], cDB);
            cDB->addMovie(clone);
        }
    }
    return cDB;
};

//return number of movies in db
long MovieDatabase::count() const {
    return movies.size();
};

```

```
//return movie as string
string MovieDatabase::movieString(Movie &mp) const {
    stringstream s2;
    s2 << mp;
    return s2.str();
}
//getters/////////

// getting pointer for the cert  and adds to vector if its a new certificate
MovieCertificate *MovieDatabase::getCert(string certStr) {
    int c = (this->certificates).size();
    int i = -1;
    while (++i < c) {
        if (*((this->certificates)[i]) == certStr) {
            return ((this->certificates)[i]);
        }
    }
    MovieCertificate *newCert = new MovieCertificate(certStr);
    (this->certificates).push_back(newCert);// adds to end of vector
    return newCert;

};

//getting pointer for genre, and add if adds it to vector if new so new genres
    can be easily added
MovieGenre *MovieDatabase::getGenre(string genreStr) {
    int d = (this->genres).size();
    int i = -1;
    while (++i < d) {
        if (*((this->genres)[i]) == genreStr) {
            return (this->genres[i]);
        }
    }
    MovieGenre *newGenre = new MovieGenre(genreStr);
    (this->genres).push_back(newGenre); //adds to end of vector
    return newGenre;
};

//gets the pointer to the certificate from the db and string (certficate)
MovieCertificate *MovieDatabaseLink::getCert(string certStr, MovieDatabase &mdb)
    {
    return mdb.getCert(certStr);
};

//gets the pointer to the genre from the db and string (genre)
MovieGenre *MovieDatabaseLink::getGenre(string genreStr, MovieDatabase &mdb) {
    return mdb.getGenre(genreStr);
};

//get movie pointer at index
Movie *MovieDatabase::getIndex(int i) const {
    if (i >= 0 && i < movies.size()) {
        return movies[i];
    } else if (i < 0) {
        return movies[0];
    }
    return movies[movies.size() - 1];
};


//true if movie in db,add it if not
bool MovieDatabase::addMovie(Movie *newMovie) {
    int i = -1;
```

```
123     int e = titles.size();
        while (++i < e) {
125         if (newMovie->getTitle() == titles[i]) {
                return false;
127         }
        }
129     (this->titles).push_back(newMovie->title); //add to title vector
        (this->movies).push_back(newMovie); //add to movies
131     return true;
    };
133
    //true if movies added from CSV
135 bool MovieDatabase::addMovie(stringstream &CSVlinestream) {
        Movie *newMovie = new Movie(this);
137     CSVlinestream >> newMovie;
        return this->addMovie(newMovie);
139 };

141
    bool compareMovie(Movie *a, Movie *b) { return (*a < *b); }
143
    bool compareMovieDesc(Movie *a, Movie *b) { return (*b < *a); }
145
    //sorts database
147 void MovieDatabase::sortDatabase(MovieDatabase::SortFields field, bool desc) {
        sortField = field;
149     sortDesc = desc;
        if (sortDesc) {
151         sort(movies.begin(), movies.end(), compareMovieDesc);
        } else {
153         sort(movies.begin(), movies.end(), compareMovie);
        }
155 };

157
    //compares films handling whether films are < == or < each other
159 bool CompareFilm::movieLT(const Movie &a, const Movie &b,
                              const MovieDatabase &db) {
161     switch (db.getSortField()) {
            case MovieDatabase::SortFields::year:
163             return a.getYear() < b.getYear();
                break;
165         case MovieDatabase::SortFields::genre:
                return a.getGenreCount() < b.getGenreCount();
167             break;
            case MovieDatabase::SortFields::certificate:
169             return a.getCert() < b.getCert();
                break;
171         case MovieDatabase::SortFields::duration:
                return a.getDuration() < b.getDuration();
173             break;
            case MovieDatabase::SortFields::average_rating:
175             return a.getAverageScore() < b.getAverageScore();
                break;
177         case MovieDatabase::SortFields::title_length:
                return a.getTitle().length() < b.getTitle().length();
179             break;
            case MovieDatabase::SortFields::title:
181         default:
                return a.getTitle() < b.getTitle();
183             break;
        }
185 }
```

```
187 bool CompareFilm::movieGT(const Movie &b, const Movie &a,
                             const MovieDatabase &db) {
189     switch (db.getSortField()) {
            case MovieDatabase::SortFields::year:
191             return a.getYear() > b.getYear();
                break;
193         case MovieDatabase::SortFields::genre:
                return a.getGenreCount() > b.getGenreCount();
195             break;
            case MovieDatabase::SortFields::certificate:
197             return a.getCert() > b.getCert();
                break;
199         case MovieDatabase::SortFields::duration:
                return a.getDuration() > b.getDuration();
201             break;
            case MovieDatabase::SortFields::average_rating:
203             return a.getAverageScore() > b.getAverageScore();
                break;
205         case MovieDatabase::SortFields::title_length:
                return a.getTitle().length() > b.getTitle().length();
207             break;
            case MovieDatabase::SortFields::title:
209         default:
                return a.getTitle() > b.getTitle();
211             break;
        }
213 }

215
    bool CompareFilm::movieEQ(const Movie &a, const Movie &b,
217                           const MovieDatabase &db) {
        switch (db.getSortField()) {
219         case MovieDatabase::SortFields::year:
                return a.getYear() == b.getYear();
221             break;
            case MovieDatabase::SortFields::genre:
223             return a.getGenreCount() == b.getGenreCount();
                break;
225         case MovieDatabase::SortFields::certificate:
                return a.getCert() == b.getCert();
227             break;
            case MovieDatabase::SortFields::duration:
229             return a.getDuration() == b.getDuration();
                break;
231         case MovieDatabase::SortFields::average_rating:
                return a.getAverageScore() == b.getAverageScore();
233             break;
            case MovieDatabase::SortFields::title_length:
235             return a.getTitle().length() == b.getTitle().length();
                break;
237         case MovieDatabase::SortFields::title:
            default:
239             return a.getTitle() == b.getTitle();
                break;
241     }
    }
```

MovieLink.h

```
1
#include <cstdlib>
3 #include <string>
#include <fstream>
5 #include <iostream>
#include <iomanip>
7 #include <vector>
#include <sstream>
9
#ifndef MOVIELINK_H
11 #define MOVIELINK_H

13 //this header was created due to issues accessing methods between films and
    database.
//I was unable to make my code more modular without my programming not working
15 //had to create classes to update the db with the parent db
class Movie;
17
class MovieDatabase;
19
using namespace std;
21
//made to update with changes to parent db
23 class MovieCertificate {
private:
25     string name;
public:
27     string toString() const {
           return name;
29     };

31     MovieCertificate(string newName) {
           name = newName;
33     };
};
35
inline bool operator==(const MovieCertificate &a, const string &b) {
37     return (a.toString()) == b;
}
39
inline bool operator!=(const MovieCertificate &a, const string &b) {
41     return !(a == b);
}
43
//made to update with changes to parent db
45 class MovieGenre {
private:
47     string name;
public:
49     string toString() const {
           return name;
51     };

53     MovieGenre(string newName) {
           name = newName;
55     };
};
57
inline bool operator==(const MovieGenre &a, const string &b) {
59     return (a.toString()) == b;
}
```

```
61
inline bool operator!=(const MovieGenre &a, const string &b) {
63      return !(a == b);
}
65
//films compared based on settings from parent db
67  //compare to a sorted parent db
class CompareFilm {
69  public:
        static bool movieLT(const Movie &a, const Movie &b, const MovieDatabase &c);
71
        static bool movieGT(const Movie &a, const Movie &b, const MovieDatabase &c);
73
        static bool movieEQ(const Movie &a, const Movie &b, const MovieDatabase &c);
75  };

77  //link movie and db for cert and genre
class MovieDatabaseLink {
79  public:
        static MovieCertificate *getCert(string certStr, MovieDatabase &mdb);
81
        static MovieGenre *getGenre(string certStr, MovieDatabase &mdb);
83  };


85
#endif /* MOVIELINK_H */
```

main.cpp

```
// author: 100214063

#include <cstdlib>
#include <string>
#include <iostream>
#include <vector>

using namespace std;

#include "MovieDatabase.h"



string wordFile = "films.txt";

int main(int argc, char **argv) {

    MovieDatabase *myDB = new MovieDatabase();

    cout << wordFile << " file added to database\n";
    ifstream movieCSV(wordFile);
    movieCSV >> *myDB;

    cout << "\nTask 1:\n";
    cout << "Sort the movies in ascending order of release date and display on
       console:\n\n";

    myDB->sortDatabase(MovieDatabase::SortFields::year, false);
    cout << "Movie list:\n\n";
    cout << *myDB;


    cout << "\n\nTask 2:\n";
    cout << "Display the third longest Film-Noir:\n\n";

    MovieGenre *filmNoirGenre = myDB->getGenre("Film-Noir");
    MovieDatabase *filmNoirFilms = myDB->genreList(filmNoirGenre);
    filmNoirFilms->sortDatabase(MovieDatabase::SortFields::duration, true);
    int posCheck = 3;
    if (filmNoirFilms->count() < posCheck) {
        cout << "Not enough films!";
    } else {
        cout << (*(filmNoirFilms->getIndex(posCheck - 1)));
        //cout << *filmNoirFilms;
    }

    cout << "\n\nTask 3:\n";
    cout << "Display the eighth most recent UNRATED movie:\n\n";
    MovieCertificate *filmCert = myDB->getCert("UNRATED");
    MovieDatabase *unratedFilm = myDB->certList(filmCert);
    unratedFilm->sortDatabase(MovieDatabase::SortFields::year, true);
    int psCheck = 8;
    if (unratedFilm->count() < psCheck) {
        cout << "Not enough films!";
    } else {
        cout << (*(unratedFilm->getIndex(psCheck - 1)));
        //cout << *unratedFilm;
    }

    cout << "\n\nTask 4:\n";
```

```
    cout << "Display the movie with the longest title:\n\n";
62
    myDB->sortDatabase(MovieDatabase::title_length, true);
64  int psCheck2 = 1;
    if (myDB->count() < psCheck2) {
66      cout << "Not enough films!";
    } else {
68      cout << (*(myDB->getIndex(psCheck2 - 1)));
        //cout << *myDB;
70  }

72
    return EXIT_SUCCESS;
74 }
```