CMP-5015Y Summative Coursework 2 - Music Database in C++

100227789 (jvf17ptu)

Wed, 1 May 2019 11:16

PDF prepared using PASS version 1.15 running on Windows 10 10.0 (amd64).

☑ I agree that by submitting a PDF generated by PASS I am confirming that I have checked the PDF and that it correctly represents my submission.

Contents

Duration.h **2**

Duration.cpp **3**

Track.h **5**

Track.cpp **6**

Album.h **7**

Album.cpp **8**

Collection.h **10**

Collection.cpp **11**

main.cpp **14**

output.txt **15**

Duration.h

```
1  #pragma once
   #include <ostream>
3
   class Duration {
5  private:
           int seconds;
7          int minutes;
           int hours;
9  public:
           Duration();
11         Duration(int seconds, int minutes, int hours);

13         int getSeconds() const;
           int getMinutes() const;
15         int getHours() const;

17         Duration operator+(const Duration& b);
           bool operator>(const Duration& b);
19         bool operator<(const Duration& b);
           bool operator==(const Duration& b);
21
           friend std::ostream& operator << (std::ostream& out, const Duration& obj)
               ;
23 };
```

Duration.cpp

```
1  #include "Duration.h"
   Duration::Duration() {
3          this->seconds = 0;
           this->minutes = 0;
5          this->hours = 0;
   }
7
   Duration::Duration(int seconds, int minutes, int hours) {
9          this->seconds = seconds;
           this->minutes = minutes;
11         this->hours = hours;
   }
13
   int Duration::getSeconds() const {
15         return this->seconds;
   }
17 int Duration::getMinutes() const {
           return this->minutes;
19 }
   int Duration::getHours() const {
21         return this->hours;
   }
23
   Duration Duration::operator+(const Duration& b) {
25         Duration duration();
           int s = this->seconds + b.seconds;
27         int m = this->minutes + b.minutes;
           int h = this->hours + b.hours;
29
           // carry over
31         if (s > 59) {
                   s -= 60;
33                 m += 1;
           }
35         if (m > 59) {
                   m -= 60;
37                 h += 1;
           }
39         return Duration(s, m, h);
   }
41
   bool Duration::operator>(const Duration& b) {
43         if (this->hours > b.getHours()) {
                   return true;
45         }
           else if (this->hours == b.getHours()){
47                 if (this->minutes > b.getMinutes()) {
                           return true;
49                 }
                   else if (this->minutes == b.getMinutes()) {
51                         if (this->seconds > b.getSeconds()) {
                                   return true;
53                         }
                   }
55         }
           return false;
57 }

59 bool Duration::operator<(const Duration& b) {
           return !(*this > b);
61 }
```

```
63 bool Duration::operator==(const Duration& b) {
        return (this->hours == b.getHours() && this->minutes == b.getMinutes() &&
            this->seconds == b.getSeconds());
65 }

67 std::ostream& operator << (std::ostream& out, const Duration& obj)
   {
69         out << obj.getHours() << ":" << obj.getMinutes() << ":" << obj.getSeconds
               ();
           return out;
71 }
```

Track.h

```
#pragma once
#include <string>
#include "Duration.h"

class Track {
private:
        std::string title;
        Duration * duration;
public:
        Track(std::string title, int seconds, int minutes, int hours);
        ~Track();

        std::string getTitle();
        Duration* getDuration();

        friend std::ostream& operator << (std::ostream& out, Track& obj);
};
```

Track.cpp

```
1  #include "Track.h"
2
3  Track::Track(std::string title, int seconds, int minutes, int hours) {
4          this->title = title;
5          this->duration = new Duration(seconds, minutes, hours);
6  }
7
8  Track::~Track() {
9          delete this->duration;
10 }
11
12 std::string Track::getTitle() {
13         return this->title;
14 }
15
16 Duration * Track::getDuration() {
17         return this->duration;
18 }
19
20 std::ostream& operator << (std::ostream& out, Track& obj)
21 {
22         out << obj.getTitle() << " - " << *obj.getDuration() << std::endl;
23         return out;
24 }
```

Album.h

```
#pragma once
#include <string>
#include <Vector>
#include "Track.h"

class Album {
private:
        std::string artist;
        std::string title;
        std::vector<Track*> * tracks;
        Duration* duration;
public:
        Album(std::string artist, std::string title);
        ~Album();

        void addTrack(std::string title, int seconds, int minutes, int hours);
        std::string getArtist();
        std::string getTitle();
        std::vector<Track*> * getTracks();
        int getNumberOfTracks();
        Duration* getDuration();
        Track* getLongestTrack();

        friend std::ostream& operator << (std::ostream& out, Album& obj);
};
```

Album.cpp

```
#include "Album.h"

Album::Album(std::string artist, std::string title) {
        tracks = new std::vector<Track*>();
        duration = new Duration();
        this->artist = artist;
        this->title = title;
}

Album::~Album() {
        for (std::vector<Track*>::iterator it = tracks->begin(); it != tracks->
            end(); it++)
        {
                delete *it;
        }
        delete this->tracks;
}

void Album::addTrack(std::string title, int seconds, int minutes, int hours) {
        // add track then update total album duration
        Track* t = new Track(title, seconds, minutes, hours);
        tracks->push_back(t);
        *(this->duration) = *(this->duration) + *(t->getDuration());
}

std::string Album::getArtist() {
        return this->artist;
}

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

std::vector<Track*>* Album::getTracks() {
        return this->tracks;
}

int Album::getNumberOfTracks() {
        return this->tracks->size();
}

Duration* Album::getDuration() {
        return duration;
}

Track* Album::getLongestTrack() {
        Track* t = nullptr;
        for (std::vector<Track*>::iterator it = tracks->begin(); it != tracks->
            end(); it++) {
                if (t == nullptr || *((*it)->getDuration()) > *(t->getDuration())
                    ) {
                        t = *it;
                }
        }
        return t;
}


std::ostream& operator << (std::ostream& out, Album& obj)
{
        out << obj.getArtist() << " - " << obj.getTitle() << std::endl;
```

```
59          return out;
}
```

Collection.h

```
#pragma once
#include <string>
#include <Map>
#include <Vector>
#include "Album.h"

class Collection {
private:
        std::map<std::string, std::vector<Album*>> * collection;
public:
        Collection();
        ~Collection();

        std::map<std::string, std::vector<Album*>>* getCollection();
        void addAlbum(std::string artist, std::string title);
        void addTrack(std::string artist, std::string albumTitle, std::string
           title, int seconds, int minutes, int hours);
        Album* getAlbum(std::string artist, std::string title);
        Duration getDuration(std::string artist);
        Album* getLargestAlbum();
        Track* getLongestTrack();
        friend std::ostream& operator << (std::ostream& out, const Collection&
           obj);
};
```

Collection.cpp

```
#include "Collection.h"

Collection::Collection() {
        collection = new std::map<std::string, std::vector<Album*>>;
}

Collection::~Collection() {
        for (std::map<std::string, std::vector<Album*>>::iterator it = collection
           ->begin(); it != collection->end(); it++)
        {
                for(std::vector<Album*>::iterator it2 = it->second.begin(); it2
                   != it->second.end(); it2++)
                        delete *it2;
        }
        delete collection;
}

std::map<std::string, std::vector<Album*>>* Collection::getCollection() {
        return collection;
}

void Collection::addAlbum(std::string artist, std::string title) {
        // add empty album
        Album* album = new Album(artist, title);
        (*collection)[artist].push_back(album);
}

void Collection::addTrack(std::string artist, std::string albumTitle, std::string
    title, int seconds, int minutes, int hours) {
        // add track to existing album
        std::map<std::string, std::vector<Album*>>::iterator it = collection->
           find(artist);
        if (it != collection->end()) {
                for (std::vector<Album*>::iterator it2 = it->second.begin(); it2
                   != it->second.end(); it2++) {
                        if ((*it2)->getTitle() == albumTitle) {
                                (*it2)->addTrack(title, seconds, minutes, hours);
                                return;
                        }
                }

        }
}

Album* Collection::getAlbum(std::string artist, std::string title) {
        std::map<std::string, std::vector<Album*>>::iterator it = collection->
           find(artist);
        if (it != collection->end()) {
                for (std::vector<Album*>::iterator it2 = it->second.begin(); it2
                   != it->second.end(); it2++) {
                        if ((*it2)->getTitle() == title) {
                                return *it2;
                        }
                }
        }
        return nullptr;
}

Duration Collection::getDuration(std::string artist) {
        // sum durations of all albums of the artist
        Duration d;
```

```
        std::map<std::string, std::vector<Album*>>::iterator it = collection->
            find(artist);
        if (it != collection->end()) {
                for (std::vector<Album*>::iterator it2 = it->second.begin(); it2
                    != it->second.end(); it2++) {
                        d = d + *((*it2)->getDuration());
                }
        }
        return d;
}

Album* Collection::getLargestAlbum() {
        Album* largestAlbum = nullptr;
        for (std::map<std::string, std::vector<Album*>>::iterator it = collection
            ->begin(); it != collection->end(); it++) {
                std::vector<Album*>* albums = &it->second;
                for (std::vector<Album*>::iterator it2 = albums->begin(); it2 !=
                    albums->end(); it2++) {
                        if (largestAlbum == nullptr || (*it2)->getNumberOfTracks
                            () > largestAlbum->getNumberOfTracks()) {
                                largestAlbum = *it2;
                        }
                }
        }
        return largestAlbum;
}

Track* Collection::getLongestTrack() {
        Track* t = nullptr, *max = nullptr;
        for (std::map<std::string, std::vector<Album*>>::iterator it = collection
            ->begin(); it != collection->end(); it++) {
                std::vector<Album*>* albums = &it->second;
                for (std::vector<Album*>::iterator it2 = albums->begin(); it2 !=
                    albums->end(); it2++) {
                        t = (*it2)->getLongestTrack();
                        if (max == nullptr || *(t->getDuration()) > *(max->
                            getDuration())) {
                                max = t;
                        }
                }
        }
        return max;
}

std::ostream& operator << (std::ostream& out, const Collection& obj)
{
        // since map iterator search the map in alphabetical order by default
        // we only need to sort the album vector of each artist
        for (std::map<std::string, std::vector<Album*>>::iterator it = obj.
            collection->begin(); it != obj.collection->end(); it++) {
                std::vector<Album*> * albums = &it->second;
                // sort arist albums using bubble sort
                for (int i = 0, n = albums->size(); i < n - 1; i++) {
                        for (int j = 0; j < n - i - 1; j++) {
                                if ((*albums)[j]->getTitle() > (*albums)[j + 1]->
                                    getTitle()) {
                                        Album* temp = (*albums)[j];
                                        (*albums)[j] = (*albums)[j + 1];
                                        (*albums)[j + 1] = temp;
                                }
                        }
                }
                for (std::vector<Album*>::iterator it2 = albums->begin(); it2 !=
```

```
                    albums->end(); it2++) {
108                     out << it->first << " - " << (*it2)->getTitle() << std::
                            endl;
                }
110
        }
112     return out;
}
```

main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include "Collection.h"
using namespace std;

int main() {
        ifstream infile("albums.txt");

        Collection myCollection;
        string line;
        string albumTitle = "", artist = "";

        // 1. populate myCollection from the provided file
        while (std::getline(infile, line))
        {
                // Album title
                if (!isdigit(line[0])) {
                        size_t found = line.find(':');
                        if (found != string::npos) {
                                albumTitle = line.substr(found + 2, line.length()
                                    - found - 2);
                                artist = line.substr(0, found - 1);
                                myCollection.addAlbum(artist, albumTitle);
                        }
                }
                else {
                        string title;
                        int s, m, h;
                        size_t found = line.find('-');
                        h = stoi(line.substr(0,1));
                        m = stoi(line.substr(2, 2));
                        s = stoi(line.substr(5, 2));
                        title = line.substr(found + 2, line.length() - found - 2)
                            ;
                        myCollection.addTrack(artist, albumTitle, title, s, m, h)
                            ;
                }
        }

        cout << "2. The entire collection sorted in alphabetical order: " << endl
            ;
        cout << myCollection << endl;

        cout << "3. The total playtime of all Pink Floyd albums: " << endl;
        cout << myCollection.getDuration("Pink Floyd") << endl << endl;

        cout << "4. The album with the largest number of tracks: " << endl;
        cout << *myCollection.getLargestAlbum() << endl;

        cout << "5. The longest track of the collection: " << endl;
        cout << *myCollection.getLongestTrack() << endl;
        return 0;
}
```

output.txt

```
2. The entire collection sorted in alphabetical order:
Blondie - Parallel Lines
Goldfrapp - Supernature
Jordi Savall & Hesperion XX - Folias and Canarios
Kraftwerk - The Man Machine
Kraftwerk - Trans Europe Express
Led Zeppelin - Led Zeppelin IV
Marillion - Script for a Jester's Tear
Miles Davis - Kind of Blue
Neil Pye - Neil's Heavy Concept Album
Nimoy & Shatner - Spaced Out
Pink Floyd - Animals
Pink Floyd - Dark Side of the Moon
Pink Floyd - Meddle
Pink Floyd - Momentary Lapse of Reason
Pink Floyd - Wish You Were Here
Pulp - Different Class
The Beatles - Rubber Soul
The Dave Brubeck Quartet - Take Five
The Jimi Hendrix Experience - Are you Experienced?

3. The total playtime of all Pink Floyd albums:
3:46:20

4. The album with the largest number of tracks:
Nimoy & Shatner - Spaced Out

5. The longest track of the collection:
 - 0:23:31


RUN SUCCESSFUL (total time: 474ms)
```