Project in C++ OOP

Hergeir Winther Lognberg
Hewi1600

1 Preamble

In this project I'm to code a "jukebox" terminal program that follows the specifications given from the teacher. I've striven to make the code as clean, readable and free of redundancies as possible. Also the one specification was to use STL and I've tried to use it as much as i deemed practical.

I'm aiming for the highest grade (we where told to tell).

2 The Code

2.1 placement

I've decided to keep all files associated with the project in the root of the project folder. There are in all 19 files containing code and 1 file containing the jukebox database. Out of them 8 are header files (1 for each class and 1 extra for some user-defined functions.

2.2 code

The header files do not contain code other than class definitions and function prototypes.

I've kept all overloadings of operators at the bottom of the relevant files to keep consistent style.

In *Jukebox.cpp* (the most substantial file) I've tried to make the placement of functions chronological to the menu switch statement orders. And functions not called directly by switches at the bottom of the file.

The first thing I'll mention here is the code in the *functions.cpp*. Here i keep some functions I've made myself for handling user input (*getInt,getLine*). And some functions to format prompts and other input in a pleasing way.

Also i chose to keep my homebrewed *toCase* function instead of using *transform* in cooperation with to-upper/lower because of the gained simplicity of use in our case.

Then there's the main function which contains nothing but the creation of a *Jukebox* object and running it's run function.

The rest of the files in the project are pairs of class header and definition files.

I really hope my comments will help to explain enough.

2.2.1 jukeBox

handles all user interaction and keeps track of it's own albums vector.

2.2.2 Menu

getMenuChoice this function takes care of everything related with user input and menu presentation. The function only accepts an integer input from user, and even then it checks whether the menuchoice exists and is enabled. If not it re-prompts user. The only non-menuItem entry it accepts is the last switch entry (the exit/return) condition.

2.2.3 Queue / Playlist

I have created a Queue that completes all requirements this assignment has set forth. It grows by 5 spaces every time it it's getting too small. And it's using indexes instead of pointers to keep track of first and last element in list. (Personally I would have used real pointers)

As It should be, user has no control over anything except:

- Appending song last in queue
- Removing first element in list and returning it to the user in the process

I chose to implement the queue such that delete is never called until the destructor is called. Every time user takes first element of queue It's immediately overwritten by the next one in queue. Because the row is replaced with it's new first entry in the first spot and last pointer moves down.

2.2.4 ”Playing” songs

The assignment told us to make the program linger a few seconds at every song before proceeding to the next. I chose the interval to be 2 seconds. I was kind of in a dilemma as there where 2 ways of doing this. I could choose between using *system(sleep)* which is kind off cross-platform (i believe the windows cmd recognizes it?) But as we only use 1 thread at any given time and where told to use STL, I thought it might be cleaner (and safer) to use what I have used:

```
#include <chrono>   // used for waiting a few seconds before continuing.
#include <thread>   // used for threadmanipulation
pop().print(1); // returns first element and deletes it
this_thread::sleep_for(chrono::seconds(2)); // cross platform c++11 for singe threaded software
```

Figure 1:

3 Building/Compiling

I’ve created a makefile for the project. It’s pretty crude, but works. Just cd into the project directory and run make. To run the program run ”make run” if you’re using linux or osx. If you’re using Windows your best bet is probably using visual studio.

4 Enviroment

I’m programming on an Arch linux 64-bit system. I’ve got the c++ compiler installed and compile using it’s g++ alias which links necessary libraries automatically. To compile i use the recommended flags: ”-std=c++11 -Wall -pedantic”. The flags let me choose to use c++11 standard and give me useful compiling warnings and errors. For editing of code i use Gedit with syntax highlighting plugin’s enabled.

Friday 17$^{\text{th}}$ March, 2017