CMSC 202 Spring 2024
Project 5 – UMBC Movie Player

Assignment: Project 5 – UMBC Movie Player

Due Date: Thursday, May 2nd on GL by 8:59pm

Value: 80 points

1. Overview

In this project, you will:

- Practice basic C++ syntax including branching structures,
- Write classes and instantiate those classes using a constructor,
- Create a templated data structure (queue or stack),
- Use simple file input, and
- Use overloaded operators to access templated data structure.

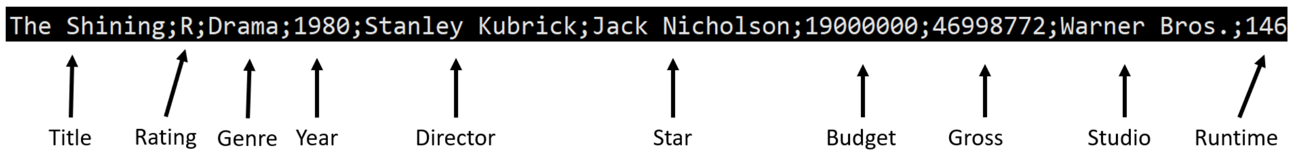
2. Background

In 2024, a movie player remains a valuable and intriguing tool in the digital landscape for various reasons. Firstly, despite the proliferation of streaming services, owning a movie player provides autonomy over one's entertainment choices. With concerns about streaming platform exclusivity, subscription costs, and internet connectivity issues, a movie player ensures uninterrupted access to favorite films and shows. Additionally, it offers a tangible collection for cinephiles, allowing them to curate personal libraries and indulge in cinematic experiences without reliance on fluctuating online catalogs. Moreover, with technological advancements, modern movie players boast enhanced features like 4K resolution, immersive sound systems, and compatibility with diverse media formats, offering a superior viewing experience. As nostalgia continues to influence consumer preferences, the movie player serves as a bridge between classic films and contemporary technology, appealing to both seasoned enthusiasts and younger generations eager to explore cinematic history.

For this project, we are going to implement a movie player that can build a playlist from thousands of available titles based on a multitude of features.

Please note: once you have completed the project, there is optional extra credit listed below. You cannot attempt the extra credit if you have not successfully implemented the swap and sort functions in the base project.

The input file for the movies looks like this:



```
The Shining;R;Drama;1980;Stanley Kubrick;Jack Nicholson;19000000;46998772;Warner Bros.;146
```

Figure 1. Input File

3. Assignment Description

Class 1 – Queue – This is a templated linked list. This is the first class you should write. It is a full data structure that allows users to **`PushBack`** (add to tail) and **`PopFront`** (remove from front). Additionally, it allows users to display, return the data in the first node, return the size, and check to see if it is empty. It also has a copy constructor and overloaded assignment operator. As it is templated, it should be able to hold just about any data type (some types of dynamic objects might gum it up).

Additionally, it comes with its own test file (**`queue_test.cpp`**) and should be tested completely before moving onto the rest of the project. You should write one function at a time starting with the constructors, **`PushBack`**, and **`Display`**. Then you can comment out all tests except test 1. Then write the copy constructor and test it. Then write the assignment operator and test it. Repeat this until all the tests have successfully passed. We will be testing just **`Queue`** by itself, and we will test all the functions individually like they are done in **`queue_test.cpp`**.

It is highly recommended that you complete everything else in the Queue class before you start to work on **`Swap`** and **`Sort`**. These are the two most challenging functions in the project. If you are short on time, skip these two functions and come back to them as they will be worth less than having a mostly working project. The sort uses a bubble sort. You can find information about bubble sorts (and more conceptual code) here: https://www.prepbytes.com/blog/linked-list/bubble-sort-for-linked-list-by-swapping-nodes/

There should be absolutely **NO** references to anything about Movie in the **`Queue`** file – it should be able to work with any data type!

You should implement this class by itself and then test it completely before using it. Do not forget how we must implement templated classes.

Class 2: MoviePlayer – This is the class that manages the master movie catalog and the user's playlist. It is created in **`proj5.cpp`**. There is one data file which will be used to dynamically allocate Movie objects that are loaded into a vector that stores all the movies (which is thousands of movies). You can display all movies available based on year and genre, add movie to the playlist, display your playlist, or sort the reading list by year (oldest to newest).

Class 3: Movie – This is the class that contains all of the information about each movie. Each movie has a title, rating, genre, year, director, star, budget, gross, studio, and runtime. There are overloaded < and > operators for sorting and << for displaying (title by director from year).

4. Requirements:

This is a list of the requirements of this application. For you to earn all the points you will need to meet all the defined requirements.

- You must follow the coding standard as defined in the CMSC 202 coding standards (found on Blackboard under course materials). This includes comments as required.
- The project must be turned in on time by the deadline listed above.
- The project must be completed in C++. You may not use any libraries or data structures that we have not learned in class. Libraries we have learned include **`<iostream>`**, **`<fstream>`**, **`<iomanip>`**, **`<vector>`**, **`<cmath>`**, **`<ctime>`**, **`<cstdlib>`**, **`<sstream>`**, and **`<string>`**. You should only use **`namespace std`**.
- Using the provided files, **`Queue.cpp, MoviePlayer.h, Movie.h, makefile, proj5_movie.txt and the proj5.cpp file`** write the program as described. (Finish Queue first though!) You must use a **`Queue`** for the read list of movies in **`MoviePlayer.h.`** You can use a vector to store the catalog of movies (which are loaded from a file).
- You may use any of the conversion functions such as **`stoi, stoul, stod`** or **`atoi`**.
- As a reminder, **`Queue.cpp`** is templated and all functions must exist in ONE file (**`Queue.cpp`**). **`MoviePlayer`** uses movies but could easily be modified to use practically any type of thing. The **`Queue.cpp`** must have no reference to Movie or MoviePlayers anywhere in it.
- You can copy the files from my directory in **`/afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj5`**.
- You may **NOT** modify the headers files (or **`Queue.cpp`**).
- Must use iterators to iterate over a vector at least once.
- The **`Queue`** must be templated with all requested functions implemented (including the copy constructor and assignment operator). Look closely at the **`makefile`** – it has additional rules for testing the **`Queue`**. There is a file named **`queue_test.cpp`** that is designed to test each of the major parts of **`Queue`**. This will help you incrementally build **`Queue`**. You can comment out everything except for the first test and then continue from there. Please test your **`Queue`** before starting the **`MoviePlayer`** aspects of this project!
- All user input must be validated. For example, if a menu allows for 1, 2, or 3 to be entered and the user enters a 4, it will re-prompt the user. However, the user is expected to always enter the correct data type. i.e. If the user is asked to enter an

integer, they will. If they are asked to enter a character, they will. You do not need to worry about checking for correct data types.

- The code must not have memory leaks, warnings, or errors.

5. Sample Input and Output

To start with the testing of the **`Queue`** file, you will be using the **`queue_test.cpp`** file and the **`makefile`**. You can alternate writing functions (starting with the constructor and **`PushBack`**) and running "make **`qtest`**" and then "make **`qtest2`**". Below is an example of the completed **`qtest2`** run when all functions have been implemented and the test runs successfully.

[redacted]

[redacted]

Once all of the **`Queue`** tests work, you should work on **`Movie`**. It is easy enough – the only tricky functions are the overloaded operators. After that, you are on to working with **`MoviePlayer.cpp.`** All user entries are in blue. Here is a sample run from that:

[redacted]

```
3



4



5
```

Below is a sample where we display all the podcasts from 2020 (which there are 100). We have omitted many for space purposes, but they should display on the screen.

```



1

2005


Animation
















5

```

Here is a full run including adding two songs and one podcast, a sort, and the display.

```
1

1999

Animation

3
```

```
2

1980

Comedy

4

2
```

```
1997

Drama

1775
```

```
2
1988
Action
```

```
634




3





4





3
```

```
5
```

There is a longer sample run available called **`proj5_sample.txt`**. There is an extra run available for the extra credit (described below) called **`proj5_extra.txt`**.

6. Compiling and Running

We have provided you the makefile for this project.

Once you have compiled using the **`makefile`**, enter the command **`make run`** or **`./proj5`** to run your program. If your executable is not proj5, you will lose points. It should look like the sample output provided above.

Because we are using dynamic memory for this project, you are required to manage any memory leaks that might be created. Anything that you use "new" for needs to be deleted. Remember, in general, for each item that is dynamically created, it should be deleted using a destructor.

One way to test to make sure that you have successfully removed any of the memory leaks is to use the **`valgrind`** command.

Since this project makes extensive use of dynamic memory, it is important that you test your program for memory leaks using **`valgrind`**:

```
valgrind ./proj5 proj5_movies.txt
```

Note: If you accidently use valgrind make run, you may end up with some memory that is still reachable. Do not test this – test using the command above where you include the input file.

If you have no memory leaks, you should see output like the following:

The important part is "in use at exit: 0 bytes 0 blocks," which tells me all the dynamic memory was deleted before the program exited. If you see anything other than "0 bytes 0 blocks" there is probably an error in one of your destructors. We will evaluate this as part of the grading for this project.

Additional information on **`valgrind`** can be found here:
http://valgrind.org/docs/manual/quick-start.html

7. Completing your Project

When you have completed your project, you can copy it into the submission folder. You can copy your files into the submission folder as many times as you like (before the due date). We will only grade what is in your submission folder.

For this project, you should submit all files to the **`proj5`** subdirectory:

`Movie.h, Movie.cpp`

`MoviePlayer.h, MoviePlayer.cpp`

`Queue.cpp`

`proj5.cpp`

For this project, you are allowed to edit **`Queue.cpp`** but no other header (.h) files.

As you should have already set up your symbolic link for this class, you can just copy your files listed above to the submission folder.

a. cd to your project 5 folder. An example might be cd **`~/202/projects/proj5`**

b. **`cp Movie.h Movie.cpp MoviePlayer.h MoviePlayer.cpp Queue.cpp proj5.cpp ~/cs202proj/proj5`**

There may be a command in your **`makefile`** that will automatically submit your code named **`make submit`**.

You can check to make sure that your files were successfully copied over to the submission directory by entering the command. Please note: You are responsible for turning in all required files.

```
ls -al ~/cs202proj/proj5
```

You can check that your program compiles and runs in the **`proj5`** directory, but please clean up any **`.o`** and executable files. Again, do not develop your code in this directory and you should not have the only copy of your program here. Uploading of any **`.gch`** or **`vgcore`** files will result in a severe penalty.

For additional information about project submissions, there is a more complete document available in Blackboard under “Course Materials” and “Project Submission.”

IMPORTANT: If you want to submit the project late (after the due date), you will need to copy your files to the appropriate late folder. If you can no longer copy the files into the proj5 folder, it is because the due date has passed. You should be able to see your proj5 files but you can no longer edit or copy the files in to your proj5 folder. (They will be read only)

- If it is 0-24 hours late, copy your files to **`~/cs202proj/proj5-late1`**
- If it is 24-48 hours late, copy your files to **`~/cs202proj/proj5-late2`**
- If it is after 48 hours late, it is too late to be submitted.

8. Extra Credit (Optional for up to 8pts)

If you have completed the base project and you have additional time, you can **optionally** complete a few additional features for extra credits. You cannot attempt the extra credit unless you have completed the entire base project including swap and sort.

****STOP**** - Turn in your base project first. You must turn in two sets of files.

1. Base project: Turn in normally (80pts)
2. Extra Credit: Create a new directory in your submission directory named **`extra_credit`** and load all files for the extra credit there. This includes all base files including the updated **`MoviePlayer.cpp`**. (up to 8pts)

DO NOT TURN IN YOUR EXTRA CREDIT AS YOUR BASE PROJECT. YOU WILL GET A ZERO FOR BOTH.

You can only attempt this after you have completed the base project and turned the base project and the extra credit in on time (no late submission for either). Additionally, office hours cannot be used to help you complete or debug the extra credit – you can only ask clarifying questions.

To earn the extra credit, you must add the functionality to the base project that allows the user to search the entire movie catalog three things:

1. They can search for a specific string and it will return any movie title or director that matches the string (case sensitive). So, in this case, “Die” would return movies like Die Hard but “die” would not.
2. They can search by year which returns any movie from a specific year.
3. They can search by minimum profit. Profit is calculated as gross – budget. The user enters a minimum profit and it shows all movies greater than that range.

The new menus should look like this:

```
5
```

```
1


Jumanji
```

You must implement all three search styles in order to earn any extra credit. You should only be editing **`MoviePlayer.h`** and **`MoviePlayer.cpp`** for the extra credit.

You cannot turn in the extra credit after the original due date. You cannot earn extra credit if you do not turn in the base project.

To turn in the extra credit, you should create a new folder in your submission folder named "**`extra_credit`**". Then copy your extra credit files to that directory.

```
mkdir ~/cs202proj/proj5/extra_credit
```

Copy all of your files to the extra_credit directory.

MAKE SURE THAT YOU HAVE SUBMITTED TWO VERSION: BASE VERSION in **`~/cs202proj/proj5`**

AND THE EXTRA CREDIT in **`~/cs202proj/proj5/extra_credit`**