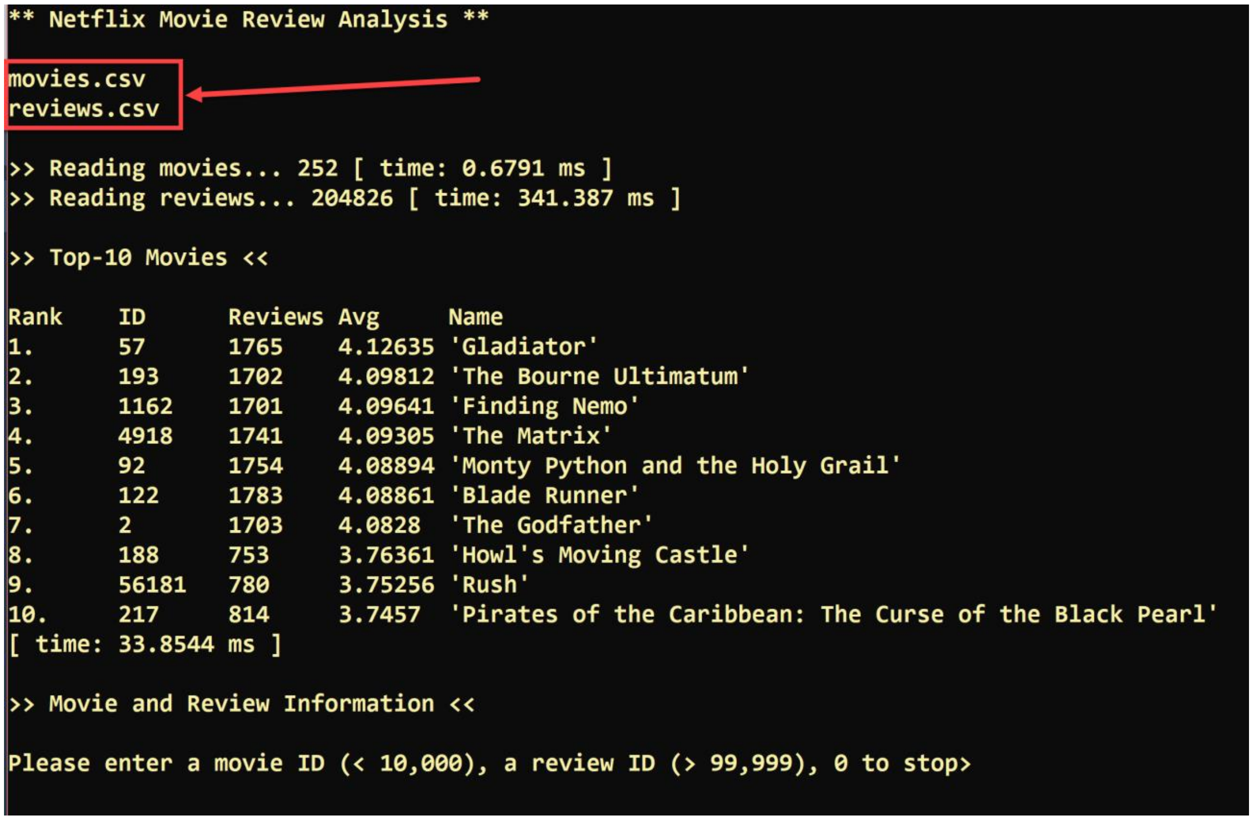
CS 341
Programming Languages
Fall 2018

Project #02: Netflix Movie Review Analysis

Complete By: Tuesday, Sept. 25th @ 11:59pm (both sections)
Assignment: modern C++ program to analyze Netflix data
Policy: Individual work only, late work up to 24 hrs after the deadline *is* accepted (see"Policy" section)
Submission: online via repl.it (exercise **P02 Netflix Analysis**)

Assignment

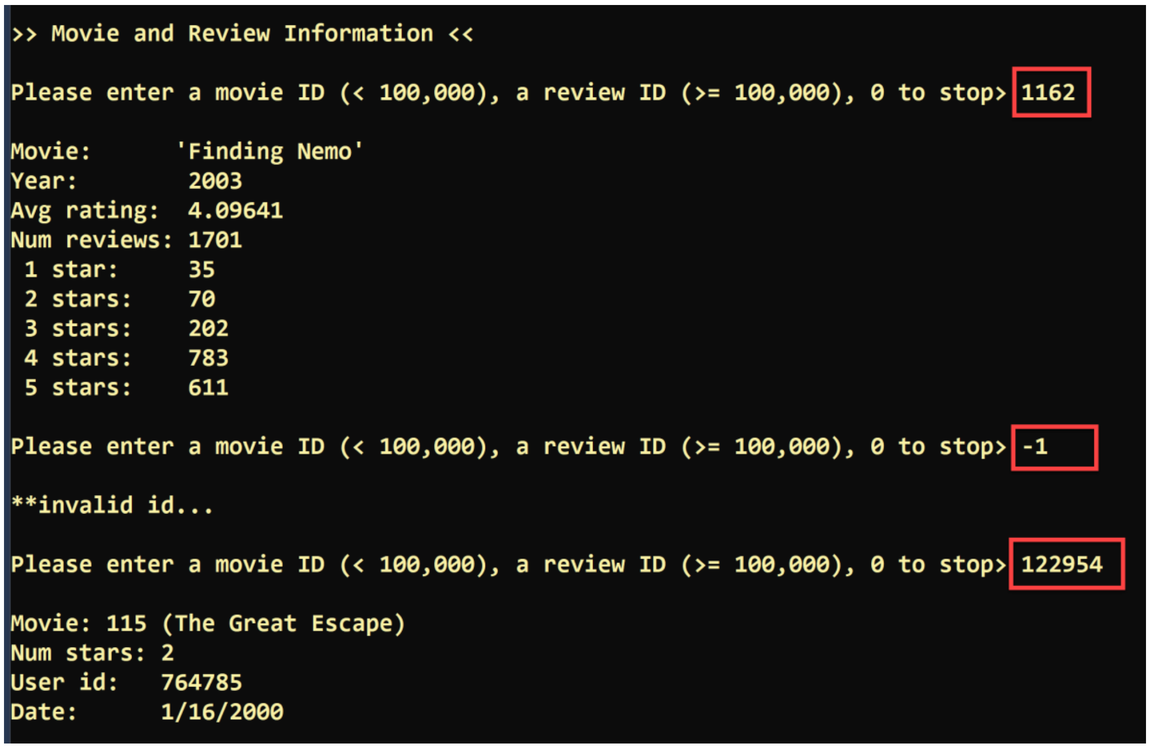
The assignment is to analyze Netflix movie reviews. There are 2 text files, one containing movie names, and one containing movie reviews (just the # of stars, 1 – 5). The program starts by inputting the filenames (movies filename then reviews filename), inputting the data from those files, and then outputting the top-10 movies based on average rating (# of stars). Here's an example screenshot:



```
** Netflix Movie Review Analysis **

movies.csv
reviews.csv

>> Reading movies... 252 [ time: 0.6791 ms ]
>> Reading reviews... 204826 [ time: 341.387 ms ]

>> Top-10 Movies <<

Rank    ID      Reviews Avg     Name
1.      57      1765    4.12635 'Gladiator'
2.      193     1702    4.09812 'The Bourne Ultimatum'
3.      1162    1701    4.09641 'Finding Nemo'
4.      4918    1741    4.09305 'The Matrix'
5.      92      1754    4.08894 'Monty Python and the Holy Grail'
6.      122     1783    4.08861 'Blade Runner'
7.      2       1703    4.0828  'The Godfather'
8.      188     753     3.76361 'Howl's Moving Castle'
9.      56181   780     3.75256 'Rush'
10.     217     814     3.7457  'Pirates of the Caribbean: The Curse of the Black Pearl'
[ time: 33.8544 ms ]

>> Movie and Review Information <<

Please enter a movie ID (< 10,000), a review ID (> 99,999), 0 to stop>
```

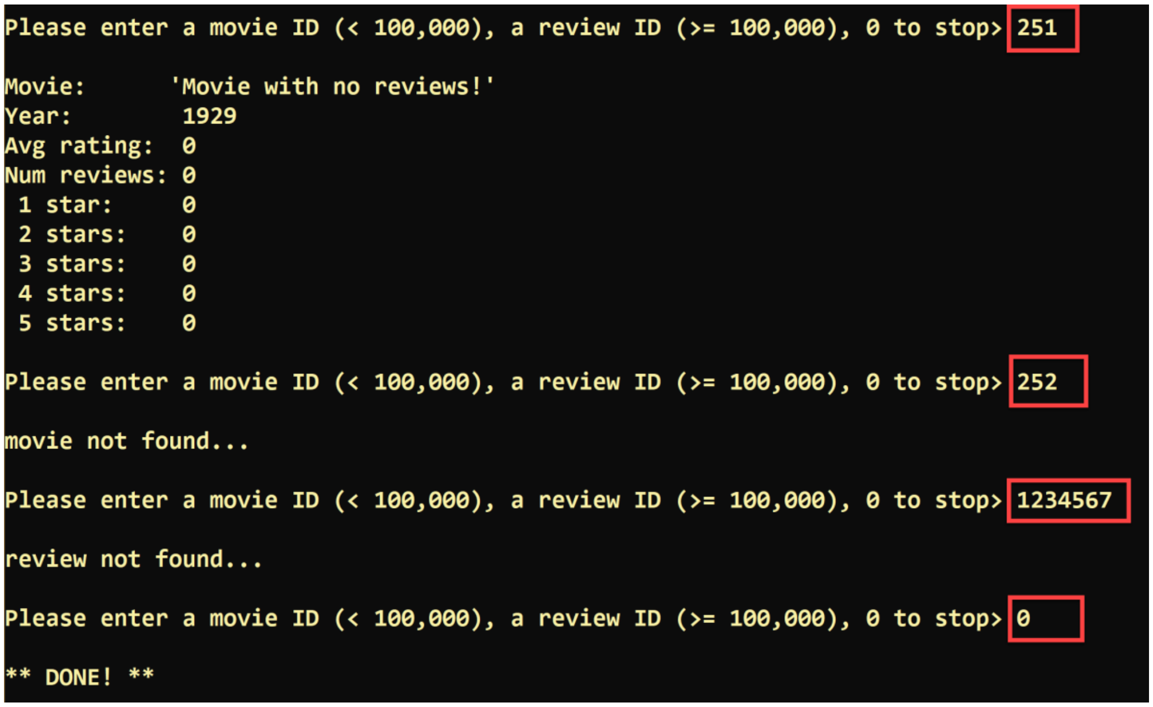
If two movies have the same average rating, the secondary sort is by movie name (ascending). The column-based output above is produced by outputting tab (\t) characters as part of the output stream. Your output should match this output as closely as possible. The times reported inside the […] are not required output, but there to judge the efficiency of your solution.

After the top-10 analysis, the second part of the program is an interactive loop that allows the user to search by movie id or review id. If the movie is found, output data about that movie (name, publication date, average rating, etc.). If the review is found, output data about that review (movie, # of stars, user id, and review date). Here's an example screenshot:



```
>> Movie and Review Information <<

Please enter a movie ID (< 100,000), a review ID (>= 100,000), 0 to stop> 1162

Movie:       'Finding Nemo'
Year:         2003
Avg rating:   4.09641
Num reviews: 1701
 1 star:      35
 2 stars:     70
 3 stars:     202
 4 stars:     783
 5 stars:     611

Please enter a movie ID (< 100,000), a review ID (>= 100,000), 0 to stop> -1

**invalid id...

Please enter a movie ID (< 100,000), a review ID (>= 100,000), 0 to stop> 122954

Movie: 115 (The Great Escape)
Num stars: 2
User id:   764785
Date:      1/16/2000
```

You may assume the user will input an integer. However, if the user enters a negative id, output an error message. Likewise, the user may enter an id for a movie or review that doesn't exist. Examples:



```
Please enter a movie ID (< 100,000), a review ID (>= 100,000), 0 to stop> 251

Movie:       'Movie with no reviews!'
Year:         1929
Avg rating:   0
Num reviews: 0
 1 star:      0
 2 stars:     0
 3 stars:     0
 4 stars:     0
 5 stars:     0

Please enter a movie ID (< 100,000), a review ID (>= 100,000), 0 to stop> 252

movie not found...

Please enter a movie ID (< 100,000), a review ID (>= 100,000), 0 to stop> 1234567

review not found...

Please enter a movie ID (< 100,000), a review ID (>= 100,000), 0 to stop> 0

** DONE! **
```

Input Files

The input files are text files in **CSV** format — i.e. comma-separated values. The file “movies.csv” defines a set of movies, in alphabetical order by movie name. The format is as follows:

```
MovieID,MovieName,PubYear
7,12 Angry Men,1957
95,2001: A Space Odyssey,1968
211,8 and a Half,1963
  .
  .
  .
129,Wild Strawberries,1957
100,Witness for the Prosecution,1957
107,Yojimbo,1961
```

The first line contains column headers, and should be ignored. The data starts on line 2, and each data line contains 3 values: a **movie ID** (a unique, positive integer < 100,000), the **movie name** (string), and the **year** the movie was **published** (integer). Note that this is just one possible input file, we may use a different file when grading (with a different # of movies and different movie data — but the file format will remain the same). The # of movies M in the file is unknown, do not assume a maximum size; you may assume that M is at least 10 (so you now there are at least 10 movies for the top-10 output).

The second input file “reviews.csv” contains movie reviews, in order by review date (with secondary sort by user id). The format is as follows:

```
ReviewID,MovieID,UserID,Rating,ReviewDate
140035,188,633086,5,1/6/2000
133994,78,1056435,4,1/6/2000
123308,199,1066384,2,1/7/2000
  .
  .
  .
147660,203,2180053,4,12/31/2005
124837,112,2188086,5,12/31/2005
289471,78,2477900,4,12/31/2005
```

The first line contains column headers, and should be ignored. The data starts on line 2, and each data line contains 5 values: the **review ID** for this review (a unique integer >= 100,000), the **movie ID** denoting the movie that was reviewed (integer), the **user id** who reviewed the movie (integer), the **rating** (the # of stars, an integer in the range 1..5, inclusive), and the **review date** (string). Note that this is just one possible input file, we may use a different file when grading. The # of reviews R in the file is unknown, do not assume a maximum size; you may assume R is at least 1. Note that some movies may not have any reviews.

Sample input files are available on the course web page under “Projects”, then “project02-files”. [*Note that the best way to download the files is *not* to click on them directly in Dropbox, but instead use the “download” button in the upper-right corner of the Dropbox web page.*] The files are being made available for each platform (Linux, Mac, and PC) so you may work on your platform of choice. Even though the files end in the file extension “.csv”, these are text files that can be opened in a text editor (e.g. Notepad) or a spreadsheet program (e.g. Excel).

Requirements

As is the case with all assignments in CS 341, how you solve the problem is just as important as simply getting the correct answers. You are required to solve the problem the "proper" way, which in this case means using modern C++ techniques: classes, objects, built-in algorithms and data structures, lambda expressions, etc. It's too easy to fall back on existing habits to just "get the assignment done."

In this assignment, your solution is required to do the following:

- Use **ifstream** to open / close the input files
- Use **Movie** and **Review** classes to store the movie and review data. For simplicity, place your class definition(s) at the top of your "main.cpp" source file. In terms of class design, all data members must be private, and accessed via public methods. Each class must also contain a parameterized constructor --- no default constructors (i.e. no parameter-less constructors that create empty objects).
- Use an efficient container class, e.g. **std::map**, to store the data. In particular, when searching for a movie or a review, this must be done using a search operation that is either O(lgN) in worst-case time complexity, or O(1) in average-case time complexity. You can still use **std::vector**, in fact consider using vector when computing the top-10 movies. But you must use at least one other type of container in your solution.
- Define and call at least **3 user-defined functions**, using efficient parameter-passing; this is in addition to the functions defined inside your classes. <u>Example</u>: write a function to input the movies from the movies file, returning the container of Movie objects; call this function from main().
- If you need to sort, sort using the built-in **std::sort** algorithm.
- **No global variables**; use parameter passing to/from your functions.
- **Total execution time < 2 minutes** in release (optimized) mode for the larger "reviews-2" file.

Not using classes? 0. Using global variables? 0. Not meeting the requirements above? 0. You get the idea --- we're looking for a modern C++ solution.

Programming Environment

To facilitate testing and program submission, we'll be using the repl.it system; see the exercise titled "**P02 Netflix Analysis**". You can work within repl.it, but we don't recomment it --- given the size of the input files, you may find that repl.it times out and kills your program, especially if your solution is inefficient. [*In fact, REPL has problems handling the large input files. So additional test cases had to be provided separately; see the REPL exercise entitled "**P02 More Tests**".*]

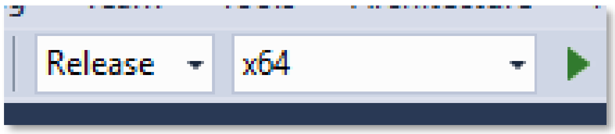
We recommend working outside of repl.it for this assignment, and only using repl.it for testing and submission. You can work within the environment of your choice: Xcode, Visual Studio, whatever you have available. In addition, we are currently testing a cloud-based Linux system from Amazon Web Services (AWS);

more details will be posted to Piazza as they become available. For those of you using a system based on g++, you may need to supply the compiler option “-std=c++11” or “-std=c++14” to enable modern C++ support.

Is your program slow?

The amount of data being processed is large enough that you may find your program running very slowly (i.e. minutes to just input the data). This could be a bad choice of algorithm or data structure, but it could also be the result of running unoptimized code.

Optimization is controlled by an option to the C++ compiler. If you are compiling with G++, include the **-O** flag as one of your compiler options (-O => “Optimize”). If you are working in Visual Studio, switch from “Debug” mode to “**Release**” mode via the drop-down on the toolbar:



Notice the drop-down shown above also displays “x64” — this configures Visual Studio to generate 64-bit code. You may need to switch from “x86” (32-bit) to “x64” (64-bit) mode in order to process larger input files.

Timing your program (optional)

If you want to time sections of your program (as shown in the screenshots), use the following code. This code outputs a duration of time in milliseconds:

```
#include <chrono>

auto beginTime = std::chrono::steady_clock::now();

// code that you want to time

auto endTime = std::chrono::steady_clock::now();
auto diff    = endTime - beginTime;

cout << " [ time: "
     << std::chrono::duration<double, std::milli>(diff).count()
     << " ms ]" << endl;
```

Electronic Submission

Before you submit, the top of each file should contain a brief overview, along with your name and other relevant info. Example:

```
//
// Netflix Movie Review Analysis
//
// <<YOUR NAME HERE>>
// U. of Illinois, Chicago
// CS341, Fall 2018
// Project #02
//
```

Your program should be readable with proper indentation and naming conventions; commenting is expected where appropriate.

The program is to be submitted via http://repl.it: see the exercise "**P02 Netflix Analysis**". The grading scheme is based on correctness, readability, commenting, and approach (including efficiency). If you pass the tests on repl.it, then you likely have a correct and efficient solution. If you fail one or more of the tests, submit anyway and the TAs will run manually outside of repl.it to determine partial credit. Note that repl.it only allows at most 30 seconds for an execution run before timing out, so if you solution is inefficient, your program will likely fail with no output on the larger "reviews-2.csv" input file. Be sure to "Submit Anyway" even if you are failing the test cases.

Policy

Late work *is* accepted. You may submit as late as 24 hours after the deadline for a penalty of 10% --- given the exam scheduled on Thursday 9/27, no submissions will be accepted after 24 hours.

Unless stated otherwise, all work submitted for grading *must* be done individually. While we encourage you to talk to your peers and learn from them (e.g. your "iClicker teammates"), this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is available here:

https://dos.uic.edu/conductforstudents.shtml .

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. It is also considered academic dishonesty if you click someone else's iClicker with the intent of answering for that student, whether for a quiz, exam, or class participation. Academic dishonesty is unacceptable, and penalties range from a letter grade drop to expulsion from the university; cases are handled via the official student conduct process described at https://dos.uic.edu/conductforstudents.shtml .