Programming Project 04 - STL

CS200 - Fall 2024-2025

Due Date: 11:55 pm, Dec 3, 2024

Lead TA(s): Namwar Ahmad Rauf

Contents

1 Overview

1.1 Plagiarism policy

- Students must not share or show program code to other students.
- Copying code from the internet is strictly prohibited.
- Any external assistance, such as discussions, must be indicated at the time of submission.
- Course staff reserves the right to hold vivas at any point during or after the semester.
- All submissions will be subjected to plagiarism detection.
- Any act of plagiarism will be reported to the Disciplinary Committee.
- Some future assignments may allow using automated code generators such as ChatGPT. However, this programming assignment does not allow using such tools.

1.2 Submission instructions

- Zip the entire folder and use the following naming convention: `<rollnumber>_PA4.zip`
  For example, if your roll number is 26100181, your zip file should be: `26100181_PA4.zip`
- All submissions must be uploaded on the respective LMS assignment tab before the deadline.
  Any other forms of submission (email, dropbox, etc) will not be accepted.

1.3 Code Quality

You are expected to follow good coding practices. We will manually look at your code and award points based on the following criteria:

- Readability: Use meaningful variable names as specified in the cpp coding standards and lectures. Assure consistent indentation. We will be looking at the following:
  - **Variables:** Use lowercase letters and separate words with under- scores. For example, my_variable
  - **Constants:** Use uppercase letters and separate words with underscores. For example, MAX_SIZE, PI
  - **Classes:** Use CamelCase (capitalize the first letter of each word without underscores). For example, MyClass, CarModel
- **Modularity and Reusability:** Break down your code into functions with single responsibilities.
- **Code Readability:** You should make sure that your code is readable. This means that your variables have meaningful names
- **Documentation** (Optional): Comment your code adequately to explain the logic and flow.
- **Error Handling:** Appropriately handle possible errors and edge cases.
- **Efficiency:** Write code that performs well and avoids unnecessary computations.

1.4 Grading distribution

Using MyString class	(4 points)
Class modularity	(3 points)
Custom template class used	(3 points)
Test 1: Pre-2000 (Small Dataset)	(28 points)
Test 2: Post-2000 (Large Dataset)	(28 points)
Test 3: Bollywood (Medium Dataset)	(28 points)
Test 4: Operator Overloading	(28 points)
Test 5: Comprehensive	(28 points)
Total	**150 points**

It is recommended that you test your data initially on the Pre-2000 dataset for convenience, as it is a relatively much smaller dataset. If your fundamental implementation is correct, your program should work fine regardless of the size of the dataset.

Marks will not be awarded for class and function modularity, as well as the usage of templates if the 5 major test cases (Test 1 - 5) do not pass.

1.5 Docker

Please note that your assignments will be tested on Docker containers. As such, it is recommended that you run your code on it at least once before submitting it. Should any errors arise on our end due to incompatibility, you will be given the chance to contest your code.

1.6 Restrictions

There are no restrictions to how you implement your program. You may import any library belonging to the C++ standard template library (STL), described here. You are expected to read up on the library's documentation and functionalities yourself.]

2 Six Degrees of Shah Rukh Khan

2.1 Background

In this project, you will be implementing a movie database. To make things more interesting, we are giving a twist to the popular "Six Degrees of Kevin Bacon" game. Here is some information about "Six Degrees of Kevin Bacon":

The game was invented in 1993 by three college friends at Albright College. The goal is to link any movie star to Kevin Bacon through a chain of movies. The minimum number of links to Kevin Bacon defines the Bacon number of the movie star. For example,
1. Jack Nicholson was in "A Few Good Men" with Kevin Bacon. Bacon number of Jack Nicholson is 1.
2. Michelle Pfeiffer was in "I Am Sam (2001)" with Sean Penn; Sean Penn was in "Mystic River (2003)" with Kevin Bacon. Therefore, Michelle Pfeiffer has a Bacon number 2.
3. For finding the Bacon number of any actor/actress, check out the link: The Oracle of Bacon

The interesting thing about this is to link any two stars through the shortest path. This is possible when complete movie databases are available (and also due to your love for writing C++ code). For this project, however, you are NOT REQUIRED to find the shortest path between two stars. Although, you may want to implement some optional class SixDegrees to do this job for extra fun and practice (not extra credit).

2.2 Objective

Impressed by your previous work on the Contiguous-inator, Dr. Doofenshmirtz has reached out to you for help in a brand new plan. It's his daughter Vanessa's birthday, and he wants to invite the most globally popular movie stars to the party. However, he claims to have been permanently banned from the internet for his past crimes and refuses to use unethical means to access it as he claims to be, and I quote, "an upstanding citizen of tri-state society".

You must develop the Khan-inator, a program that takes scattered and incomplete data related to movies and actors to develop a comprehensive movie database for Dr. Doof.

2.3 Data provided

You have been provided with data gathered from various groups of people. If a group of people is asked to link two movie stars, It is difficult for each of them to come up with a link on an individual basis. Therefore, all the people in a group consolidate their memories to increase the chances of finding a link. This allows you to gather large amounts of data without accessing the web. Each player writes whatever he or she remembers and you, the programmer with a burning passion for C++, write a program that consolidates whatever they remember. This is all possible because you know how to store unique identifiers without repeating them.

A few important points to remember regarding the data provided is:

- Each file reflects the memory of a group of people.
- The files will be variable in size according to people's memory capabilities.
- Each file carries a variable number of records, but each record follows a set format:
  - The very first line in the record is the name of the movie followed by the year.

  - The next line(s) before an empty line indicates the stars that a particular person could remember as appearing in that movie. Each star is written on a separate line.
  - Each record is separated by one or multiple lines that do not contain whitespace character(s).

- Some people could only remember the name of a movie, but not the names of any actors and actresses in that movie.
- There aren't any people who remembered the name of a star without remembering the name of the movie in which they appeared.
- There aren't any people who remember the name of a movie without remembering the year in which it was released.

Wow, this makes life simpler. Who would have guessed that Jackie Brown is actually a movie and not a star?

2.4 Sample Code & Approach

You have been provided with code for a sample implementation of a wrapper class called MyStrVec around a list data structure. This code only serves as an example; it is in no way or form the final program or even near the final program that you must implement for building the database. It only serves as a guideline on how to incorporate STL containers when structuring your classes and data structures.

The first four parts of the grading breakdown deal with how you implement your code. We have provided a 'MyString' class in the boiler code, which the first point refers to. Moreover, regarding class modularity, you will perhaps use something similar to the MyStrVec class when designing your classes. Think of the fact that an internal data structure is needed (ObjectMap would be a more fitting name for such a class) for modularity and templating. You will decide the internal data storage and representation carefully on your own. Make sure to account for repetition, as repetition would be extremely common due to a large number of data gathered from various people, resulting in a lot of common movies and stars in the data files.

2.5 Recommendations

There are many ways to implement this assignment. However, there are some recommendations that I would make which would substantially make it easier. The assignment relies heavily on using maps to track relations between keys and values. The STL library contains various types of maps, one of which you will study in class (unordered map). I would recommend looking into the ordered map for this assignment, as many functionalities require sorted outputs. Moreover, the sorting function from the C++ standard library will make your lives simpler.

2.6 Required Functionalities

The 'MoviesDB' class will be the class exposed to the testing library. Behind the scenes, you may use any form of implementation that contains the STL library functionality (the algorithms library will also be very useful). Your 'MoviesDB' class must implement the following functionalities, as they will be used to test your code.

- **void process_file(const string filename)**
  Take the name of a data file as input and process it to build your movie database. The argument passed to this function will be the filename (file path inclusive). This function will be used by the testing interface to initialize your database.

- **void print_cast(string movie, const int criteria)**
  Print the cast of the specified movie according to the criteria. The ordering criteria is either ALPHABETICAL or MOST_POPULAR which would be declared as static const ints in your class. The former means you will print the results by the alphabetical order of names, and the latter means that you will arrange the cast in decreasing order of them being remembered as belonging to that movie.

  For example, 7 people remembered that there is a movie named "Heat". 6 of them remembered that Robert De Niro appeared in "Heat", 5 remembered that Al Pacino was also in there, while only two people could remember that the movie also starred Val Kilmer. Therefore a call to print_cast("Heat", MOST_POPULAR) would print as follows.

  Heat 1995
  Robert De Niro
  Al Pacino
  Val Kilmer

  Similarly, a call to print_cast("Heat", ALPHABETICAL) means printing will be done in the order where Al Pacino appears on one line, Robert De Niro on the next line and Val Kilmer on the line after that, as shown below

  Heat 1995
  Al Pacino
  Robert De Niro
  Val Kilmer

  In the MOST_POPULAR criteria, for cases where two or more stars have the same popularity, those specific stars must be printed in ascending alphabetical order while maintaining their relative popularity order.

  Moreover, if there are multiple movies with the same name, print them all, but they must be relatively ordered according to their year of release, in ascending order.

  If the movie queried does not exist in your database, print "<movie name> does not exist in the database". If a movie exists in the database but has no associated cast information, print "<movie name> <movie year> has no cast in the database".

- **void print_movies(string star, const int criteria)**
  Print all the movies in which the movie star has appeared according to the criteria. For example, 8

people remembered Aamir Khan as appearing in "3 Idiots", 6 remembered him in "Dangal", 5 remembered that he starred in "Taare Zameen Par", while only 2 remembered him as appearing in "Ghajini". So, a call to print_movies("Aamir Khan", MOST_POPULAR) will print as follows.

Aamir Khan
3 Idiots
Dangal
Taare Zameen Par
Ghajini

Similarly, a call to print_movies("Aamir Khan", ALPHABETICAL) would print.

Aamir Khan
3 Idiots
Dangal
Ghajini
Taare Zameen Par

In the MOST_POPULAR criteria, for cases where two or more movies have the same popularity, those specific movies must be printed in ascending alphabetical order while maintaining their relative popularity order.

If the star queried does not exist in your database, print "<star name> does not exist in the database".

- **void print_costars(string star)**
  List all the costars of the given movie star in a sorted order alphabetically. Also, print a list of movies in which the two have co-starred, ordered by year.

  For example, print_costars("Brad Pitt") would result in:

  Bradd Pitt appeared with 3 different stars!
  - Appeared with Anthony Hopkins in:
    - o Legends of the Fall (1994)
    - o Meet Joe Black (1998)
  - Appeared with Julia Roberts in:
    - o The Mexican (2001)
    - o Confessions of a Dangerous Mind (2002)
  - Appeared with Val Kilmer in:
    - o True Romance (1993)

  The 'o' with each sub-item (each movie name) is just the lowercase alphabet 'o'. Obviously, the above result is just an example to clarify the print formatting. Your output would be different.

- **vector<string> get_most_popular_movies()**
  Return a vector (of simple C++ strings) containing the names of the movies that highest number of people have remembered. The movie names must be sorted in alphabetical order. For example, 5 people remembered movie 'A', 4 remembered movie 'B', 5 remembered movie 'C', and 2 remembered movie 'D'. THe highest number of people remembering a given movie is 5, which is both movie 'A' and 'C'. So, 'A' and 'C' will be returned (sorted).

- **vector<string> get_most_popular_stars()**
  Return a vector (of simple C++ strings) containing names of the stars whose computed value of popularity is the highest. The vector will contain all those stars with the highest popularity, i.e. tying for the highest popularity. The star names must be sorted in alphabetical order.

  Popularity here does not mean the number of times this star has appeared in the data files. Instead, the popularity of stars is to be computed by taking the ratio of the total number of people who remembered this star as appearing in any movie to the total number of people who actually wrote the name(s) of those movies. For example, let's assume 7 people wrote the name of the movie "Heat," 6 remembered Robert De Niro as appearing in "Heat", and 11 people also remembered the name of the movie "Jackie Brown", and 4 remembered Robert De Niro appearing in that movie. Then, assuming no one else remembered him as appearing in any other movie, the popularity of Robert Di Niro is 10/18 (i.e. (6+4)/(7+11)).

- **MoviesDB operator+=(const MoviesDB&)**
  Overload the '+=' operator to be able to merge together two movie databases (Note: this will not be simple concatenation). For example, db1 += db2 would result in db1 (database 1) containing both the data of db1 and db2, whereas db2 would remain unaffected.

3 Testing

You have been provided datasets divided into three broad categories: Pre-2000 Hollywood movies (small dataset), Post-2000 Hollywood movies (large dataset), and Bollywood movies (medium dataset). When developing your program, it is highly recommended that you do initial testing using the Pre-2000 database (Test 1) to be able to conveniently understand and debug your outputs.

To test your program, run the test.cpp file present in the test directory.

```
g++ test/test.cpp & ./a.out
```