CPSC 2430 Data Structures

Homework Assignments #5

Assigned: Friday, 11/12/2021

Due: 11:59pm, Monday, 11/22/2021

1. Group Project or Individual Project: You Pick One!

You can take this assignment as a group project (a group size of 2) or individual project.

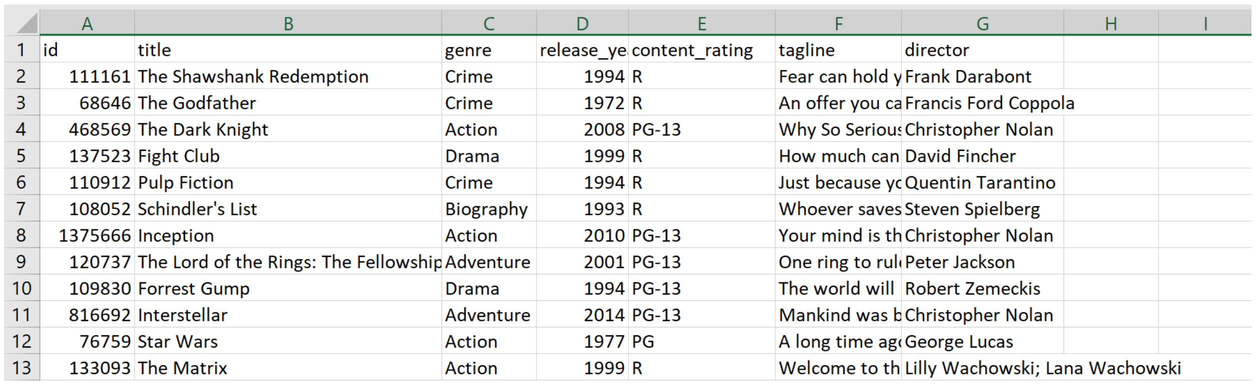
If you take it as a group project, you need additionally submit a README text file which specifies the team members and their respective contributions. Only one submission is required for each group and either team member can submit it on behalf of that team.

2. About Deadline Extension

To complete this programming assignment successfully, checkpoints are suggested in Section 5. If you plan to ask for deadline extension, prior communication is required. Also, you need to show evidence of progress you have made at the time of request based on the checkpoints. Please refer to Syllabus for details.

3. Problem

You friend, Bob, is a big fan of movies. He wants to create an in-memory movie search system that allows efficient query based on movie titles. The movie information is stored in a movies.csv file. Each movie has the following fields (separated by a comma ‘,’), including movie title, genre, content_rating, tagline, director, id and release_year. Some examples of the movie information are shown below.



	A	B	C	D	E	F	G	H	I
1	id	title	genre	release_ye	content_rating	tagline	director		
2	111161	The Shawshank Redemption	Crime	1994	R	Fear can hold y	Frank Darabont		
3	68646	The Godfather	Crime	1972	R	An offer you ca	Francis Ford Coppola		
4	468569	The Dark Knight	Action	2008	PG-13	Why So Serious	Christopher Nolan		
5	137523	Fight Club	Drama	1999	R	How much can	David Fincher		
6	110912	Pulp Fiction	Crime	1994	R	Just because yc	Quentin Tarantino		
7	108052	Schindler's List	Biography	1993	R	Whoever saves	Steven Spielberg		
8	1375666	Inception	Action	2010	PG-13	Your mind is th	Christopher Nolan		
9	120737	The Lord of the Rings: The Fellowship	Adventure	2001	PG-13	One ring to rul	Peter Jackson		
10	109830	Forrest Gump	Drama	1994	PG-13	The world will	Robert Zemeckis		
11	816692	Interstellar	Adventure	2014	PG-13	Mankind was b	Christopher Nolan		
12	76759	Star Wars	Action	1977	PG	A long time ag	George Lucas		
13	133093	The Matrix	Action	1999	R	Welcome to th	Lilly Wachowski; Lana Wachowski		

Bob comes to you for help. You convince him that the efficient movie search system can be implemented by using hashing. Specifically, you and Bob decide to design and implement a C++ class named `MovieDB` for this movie search system. A data entry stored in `MovieDB` is shown below.

```
class MovieEntry {
    private:
       string title;
       string genre;
       string contentRating;
       string tagline;
       string director;
       long id;
       int release_year;
    public:
       void setTitle(string t);
       void setGenre(string g);
       void setContentRating(string rate);
       void setTagline(string tline);
       void setDirector(string dir);
       void setId(long id);
       void setYear(int year);
       string getTitle() const;
       string getGenre() const;
       string getContentRating() const;
       string getTagline() const;
       string getDirector() const;
       long getId() const;
       int getYear() const;
};
```

The `MovieDB` needs to support the following operations:

(1) bool put(const MovieEntry &m);
The put() inserts a movie into a hash table if it does not exist. It returns true if insertion is successful and false otherwise.
(2) MovieEntry *get(const string &title);
The get() retrieves a movie (MovieEntry) with the specified title. It returns NULL if no such a movie exists.
(3) void remove(const string &title);
The remove() removes the movie with the specified title. Do not forget to deallocate dynamic memory used by the MovieEntry object.

In addition, the `MovieDB` should have the following functions: default constructor and destructor. Copy constructor and assignment operator are not required for this assignment.

The evil is in the details. To make your MovieDB practically useful, you need to consider four important issues regarding hashing.

(1) hash table size: an initial hash table size, which should be a prime number close to 2^b. Your hash table size can start with 17, for example.
(2) Your hashing algorithm is load aware. When the load factor exceeds 0.7, you should double the hash table size and rehash all existing movies. Also, the hash table dynamically halves its size when the load factor drops to 0.25 or below, but it never gets its size smaller than 7.
(3) Hashing function: You use the following hash function to hash a movie's title into an entry in the hash table. We do not use movie id as the key on purpose for this project.

```
int hash(string s) {
        unsigned int sum = 0;
        for each character c in s {
                sum = (sum*31 + c) % m;    // m is the hash table size
        }
        return sum;
}
```

(4) Collision resolution: separate chaining will be used.

There are some other important design issues:

(1) What data structure should be used for the hash table for MovieEntry? A static array or dynamic array?
Answer: You probably need a dynamic array due to rehashing caused by expanding/shrinking of the hash table.
(2) What data members are in MovieDB?
Answer: the dynamic array and its capacity, # of movies stored, etc.
(3) What should be stored in hash table entries?
Answer: header pointers (Node*) to linked list of a C++ Node as shown below.
```
struct Node {
    MovieEntry * movie_record;
    Node* next;
};
```
(4) What are the relationships between C++ classes MovieEntry, MovieDB and Node?
Answer: MovieEntry and MovieDB can be treated as separate classes. You can declare Node in private section of MovieDB.

If you cannot understand the above questions and answers, do not start your coding.

4. The Client Program

You need to implement a client program named Client.cpp to test your MovieDB.

Specifically, your client program needs to perform the following operation in order.

(1) Create a MovieDB object named db using the default constructor.
Read in all the movies from a movies.csv file and add them to db by calling put().
If you want to learn how to read a CSV file, Please refer to Section 2 "Read a particular record" at https://www.geeksforgeeks.org/csv-file-management-using-c/ for the uses of ifstream/fstream,

getline, and stringstream. Simply copying the code won't work for your project. You can also adapt the code from Programming assignment #4 to read a CSV file.

(2) There are two movie data files are provided. You can assume that movie titles are unique in both files. One .csv file is small data set and the other is large data set. Your program can assume the movie data file is movies.csv. When you test it with different movie data files, you can simply rename each file to movies.csv.

A secondary purpose of this step is to examine if the hash table doubles its size when the load factor exceeds 0.7 and rehashing is performed afterwards. Whenever the hash table increases, please print a message like:

Hash table increases from xxx to xxx.

(3) Use a loop to let users to input a movie title (be cautious: spaces are valid for a movie title) and use the movie title to search db. If the movie is in db, you need to display a movie as is:
ID: xxxxx
Title: xxxx
Genre: xxxx
Content Rating: xxxx
Tagline: xxxx
Director: xxxx
Release Year: xxxx

Do not forget to deallocate the dynamic memory after the movie info is displayed above.

If the movie does not exist in db, then simply display a message "NOT FOUND!"

(4) Use a loop to let users to input a movie title (be cautious: spaces are valid for a movie title) and use the movie title to delete the corresponding entry in db. If the movie exists and is deleted successfully, display a message "Success!". Otherwise, display a message "NOT FOUND!"

A secondary purpose of this step is to examine if the hash table shrinks when the load factor drops to 0.25 and rehashing is performed afterwards. Whenever the hash table shrinks, please print a message like:

Hash table shrinks from xxx to xxx.

(5) Repeat (3). The main purpose here is to check if the remove() operations are successfully performed in (4).

5. Checkpoints

Date	Checklist
11/12 – 11/16	(1) Read data from .csv (2) Implement C++ class MovieEntry (3) MovieDB::put() (4) MovieDB::get() In this phase, you can use a relatively large hash table without considering expanding/shrinking of the table and rehashing. You should be able to read data from movies.csv files and insert them into the hash table. You should be able to search movies by get()
11/17 – 11/19	(1) MovieDB::remove() (2) You should use the initial hash table size of 17, and then dynamically expand/shrink the hash table as needed. If the table is expanded/shrunk, then you need to do rehashing.
11/20 – 11/22	Your program should be able to work with both small and large data files. Testing. Remove all debugging & testing messages. Submit your assignment on cs1.

6. Grading Criteria

`Label`	`Notes`
[1] Submission (1 pt)	All required files are submitted.
[2] Makefile (1 pt)	Your Makefile can compile the code and generate the executable file. README for a group assignment submission.
[3] Presentation (1 pt)	Clean, well-commented code. No unsolicited output messages such as testing & debugging messages. No memory leak.
[4] Functionality (13 pts)	[1] All steps 1-5 in client.cpp must be implemented properly: 1 pt [2] Default constructor: 1pt [3] Destructor: 1pt [4] put(): supports 2 operations (a) insert the movie and (b) increase the hash table size & rehash when needed to receive full credits: 3 pts [5] get(): 3 pts [6] remove():supports 2 operations (a) remove the movie with the specified title and (b) shrink the hash table & rehash when needed to receive full credits: 3 pts [7] Messages must be displayed as specified when the hash table expands or shrinks: 1 pt
[5] Bonus points (2 pts)	If your program works perfectly with a large data file (you rename it to movies.csv when you test it), you'll receive a bonus of 2 pts.
[6] Overriding policy	If the code cannot be compiled or executed (segmentation faults instantly, for instance), it results in zero point.
[7] Late submission	Please refer to the Syllabus for details.

7. Submission

You need to submit the following files:

- movieentry.h: header file for MovieEntry class. You need to include Redundant Declaration in the header file.
- movieentry.cpp: implementation file for MovieEntry class.
- moviedb.h: header file for MovieDB class. You need to include Redundant Declaration in the header file. You need include movieentry.h in this header file.
- moviedb.cpp: implementation file for MovieDB class.
- client.cpp: a client program to test the MovieDB class, as described in Section 2.
- Makefile
- README (only required for group project)

Before submission, you should ensure your program has been compiled and tested (extensively). Your assignment receives zero if your code cannot be compiled and executed.

You can submit your program multiple times before the deadline. The last submission will be used for grading.

To submit your assignment, you should follow two steps below (assuming your files are on cs1.seattleu.edu):

1). Wrap all your files into a package, named ***hw5.tar***

```
tar  -cvf  hw5.tar  movieentry.h  movieentry.cpp moviedb.h  moviedb.cpp  client.cpp  Makefile
```

Or

```
tar  -cvf  hw5.tar  movieentry.h  movieentry.cpp  moviedb.h  moviedb.cpp  client.cpp  Makefile  README
```

2). Submit your newly generated package ***hw4.tar*** as the third programming assignment ***p5***

```
/home/fac/zhuy/zhuy/class/SubmitHW/submit2430    p5    hw5.tar
```

The message similar to the following one will be displayed if your submission is successful.

```
=====================Copyright(C)Yingwu Zhu=========================
Sun Mar 28 23:01:52 PDT 2021
Welcome testzhuy!
You are submitting hello.cpp for assignment p1.
Transferring file.....
Congrats! You have successfully submitted your assigment! Thank you!
Email: zhuy@seattleu.edu

==================================================================
```