Movies Database

Team 6

Group Members:
Abdullah Malik

Arnesh Regmi(leader)

Ulysses Morgan

Sunho Kim

Introduction: Our project, as the name suggests, is a Database consisting of movies, the database includes the primary key, movie title(secondary key), its release date, the country of production, its director, and its age rating. The program will require an input file and will get the data taken from the file and input it into an online database, where it will create a hash table, sorted based on the primary key and a tree, sorted based on the secondary key. The program will display the sorted version of the list. The program will allow for new inputs of movies as well as the option to delete a movie. One can also search for movies based on their primary key or their secondary key, in the case of the secondary key search all movies with the same key will be displayed. Finally, the program will write the data onto a file and exit.

Overview:

Application Data: **Movie** with the following member variables:

1. Title+counter – primary key (string, unique)
2. Title – secondary key (string, not unique)
3. Release date(int)
4. Language(string)
5. Year(int)
6. Director(string)
7. Is Adult(boolean)

Choice of application data: Movies, because it has a large set of options with each movie having many secondary keys to choose from.

Choice of primary key: We chose to create an ID, a combination of a counter and the title as a string.

Choice of secondary key: We chose the title as the secondary key

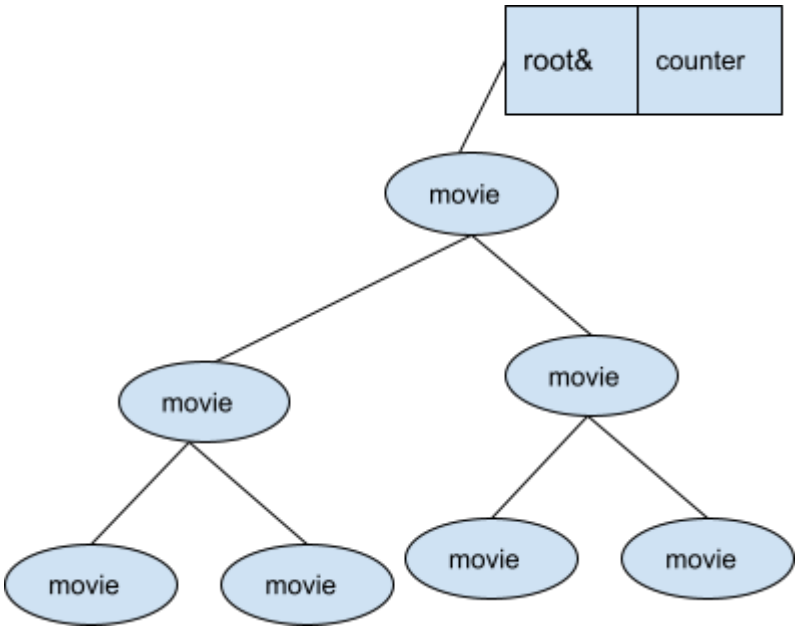
GitHub repository: We created a Github repository and every member has access to it. This is for working cohesion among group members

Data Structure Designs

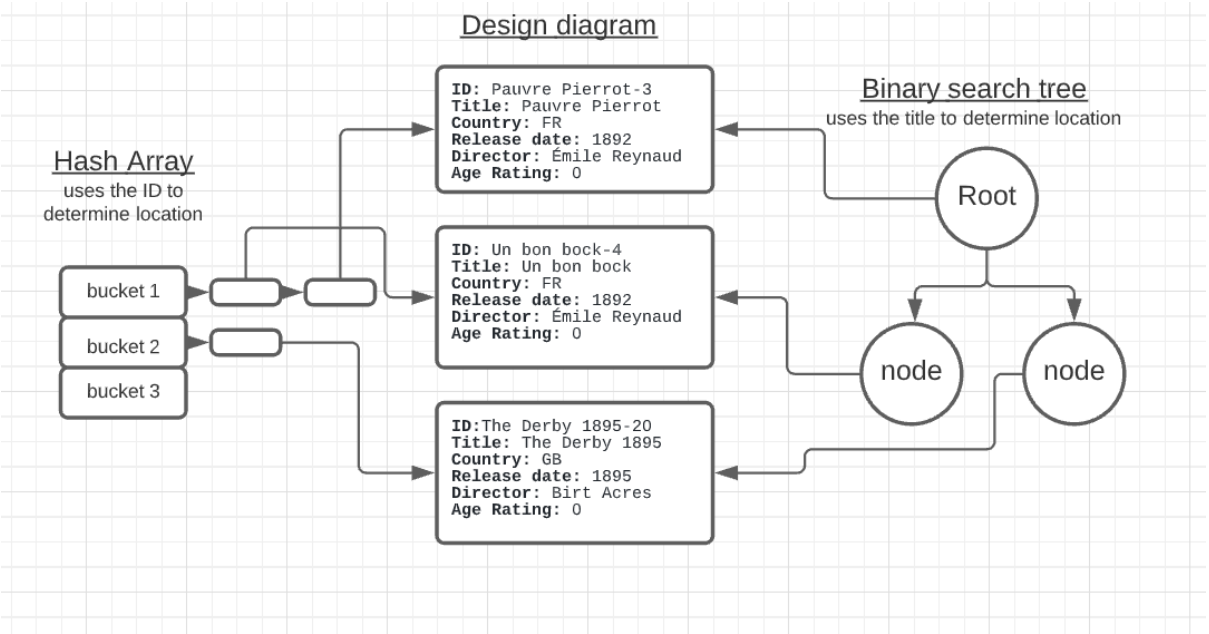
Hash function design:

Bucket 0	Bucket 1	Bucket 2	Bucket 3	Bucket 4	Bucket 5
Id:5	Id:3	Id:4	Id:1	Id:1	
↓	↓	↓	↓		
Id:7			Id:2		
↓					
Id:8					

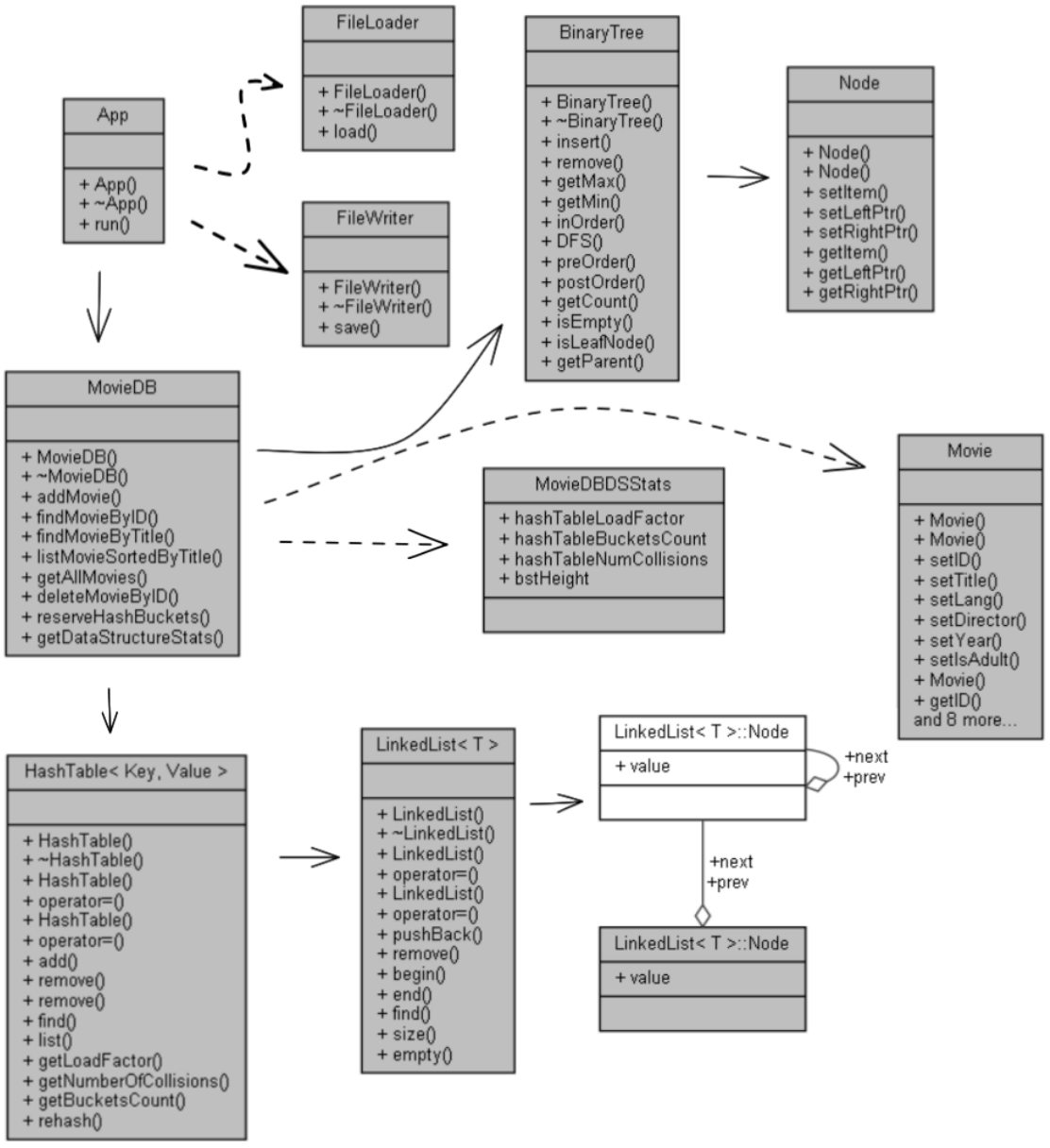
BST design



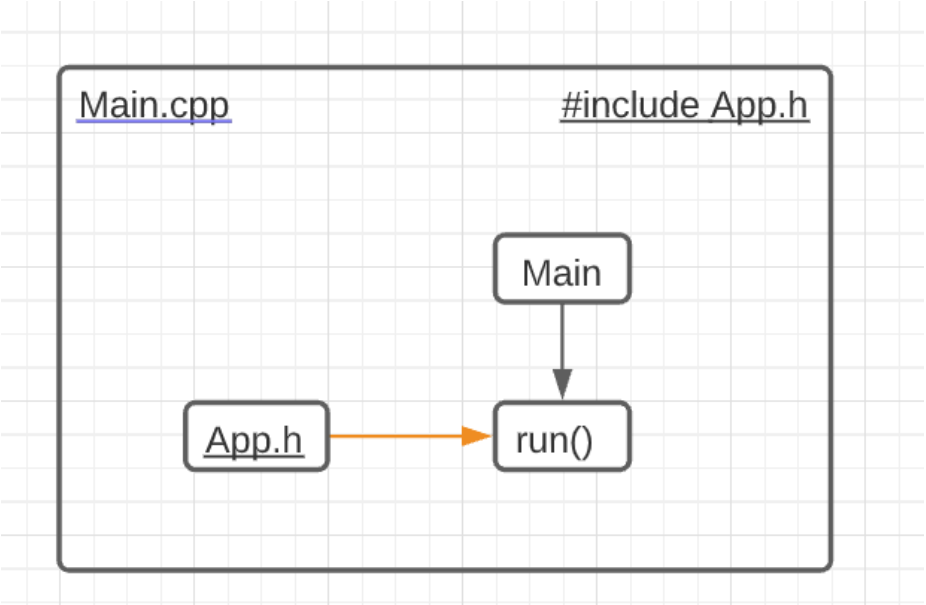
overall structure diagram:

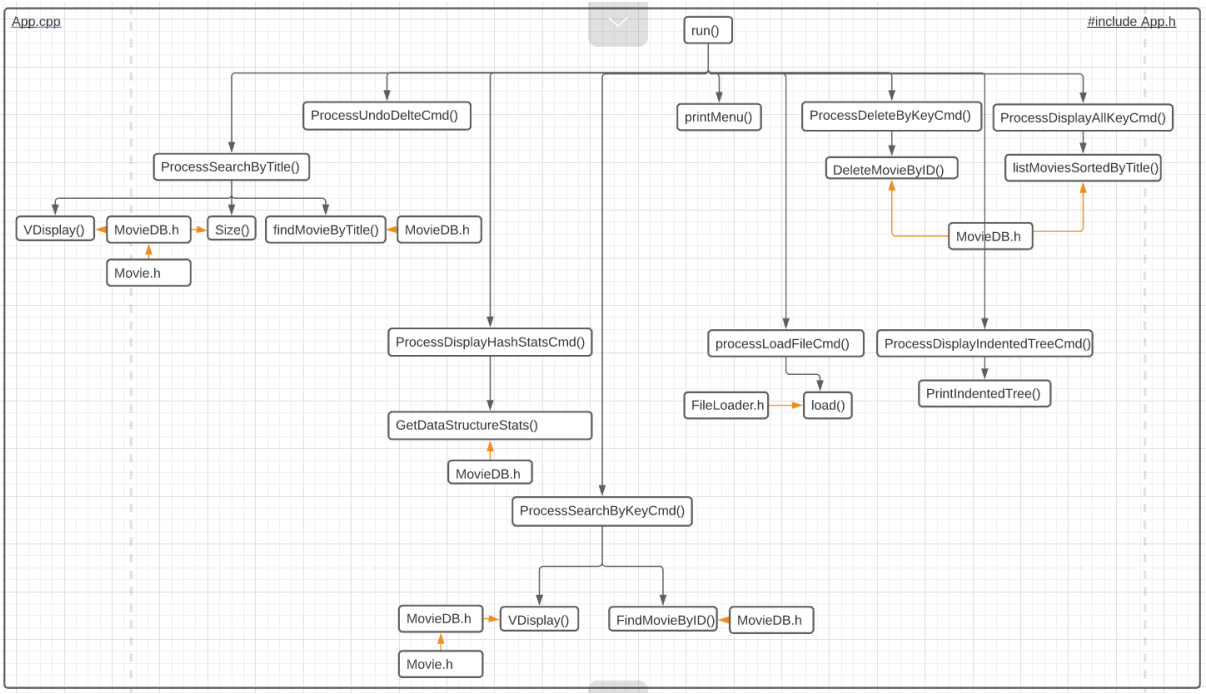


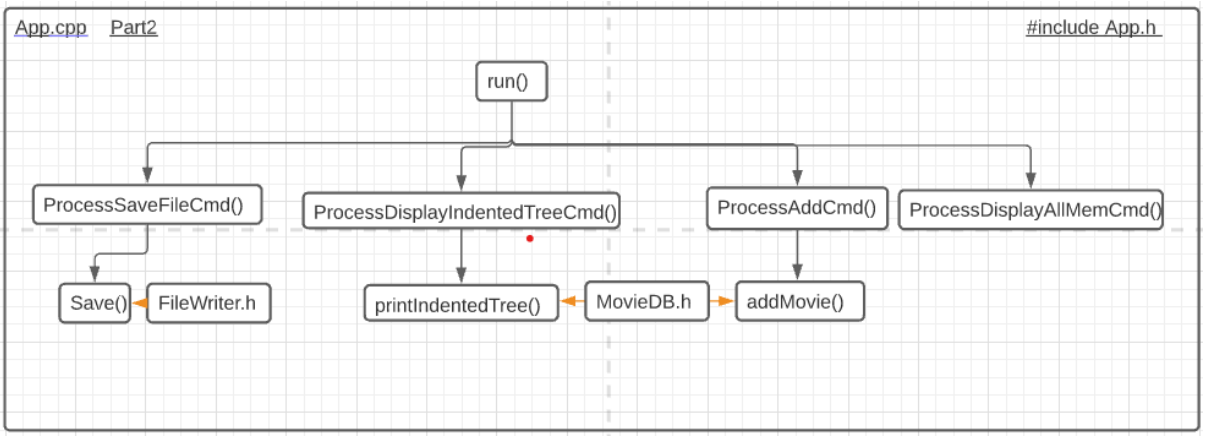
Class Diagrams:



structure chart:







Each member's task

Abdullah Malik	- File/io Writing the code in charge reading the data from the given movie file than writing the results onto a second file.
Arnesh Regmi	- screen output/coordination Screen output, essentially everything inside of main, and being in charge of the code that results in an output of somesort.
Ulysses Morgan	- finish BST Write the algorithm responsible for allowing the BST to work, inserting traversing and what not
Sunho Kim	- hash list algorithm gets the Primary key to Hash and input into a hash list to than be able to access for appropriate usage.
Everyone	- documentation and presentation Document the progression finish the presentation pdf and upload appropriate tables as well as finish the presentation poster.

Hash function raw code:

```cpp
// Hash movie id using fnv hash function
const static uint64_t fnvSeed = 14695981039346656037ull;
const static uint64_t fnvPrime = 1099511628211ull;
uint64_t movieIDHasher(const MovieID& movieID) {
    uint64_t result = fnvSeed;
    size_t alignedSize = movieID.size() - movieID.size() % 8;
    // Hash each 8-byte chunk of the movie id
    for (size_t i = 0; i < alignedSize; i+=8) {
        result ^= *reinterpret_cast<const uint64_t*>(movieID.data() + i);
        result *= fnvPrime;
    }
    // Hash unaligned part
    for (size_t i = alignedSize; i < movieID.size(); ++i) {
        result ^= movieID[i];
        result *= fnvPrime;
    }
    return result;
}
```

Our hash function (Fowler–Noll–Vo) was the default hash function used by python before, and it's known for its simplicity and fastness. It uses bitwise XOR operation to "mix" the hash, which is a really cheap computation and changes the bit pattern abruptly.

Collision resolution method:

Separate chaining method:
We have a linked list per each bucket. When a collision occurs, we add a synonym after the tail of the linked list of the corresponding bucket.