Brandon Cruz-Youll
Trinity Godwin
Yoshimasa Iwano
Armando Partida
Victor Uong

main.cpp
1. void main(std::string select, MovieHashMap_NS::MovieHashMap movieMap, MovieHashMap_NS::MovieHashMap actorMap)
a. input string select, and a movie multimap and an actor multimap
b. output nothing, function is void
c. sample input:
the string is 'genre', 'actor' or 'random' while the multimaps are always the same
d. pseudocode
if select is 'genre'
string to hold genre
ignore whatever is in cout buffer
printout "What genre would you like recommendations?"
get users genre
initialize vector of movies using movieMap -> getMovies(genre)
for top 5 movies
print out "Title: " and movie title
if select is 'actor'
string to hold actor
ignore whatever is in cout buffer
printout "What actore would you like recommendations?"
get users actor
initialize vector of movies using movieMap -> getMovies(actor)
for top 5 movies
print out "Title: " and movie title with genre
if selected random movie
select random movie based on random number generator and the total number of movies

e. given the use of multimaps, this should be logorithmic complexity.

2. int main()
a. input none
b. output 0 if all goes well
c. sample input: n/a
d. pseudocode
create new movieMap
get the genre dataset
create new actorMap
get the credits dataset

iterate through dataset
get title
get genre
get rating
put into Movie object
put Movie object into movieMap with key genre

iterate through dataset
get title

Brandon Cruz-Youll
Trinity Godwin
Yoshimasa Iwano
Armando Partida
Victor Uong

get genre
get cast
create movie object
for each cast member
put movie object into moviemap with key actor

while loop for menu
print out Movie Recommendations
print out instructions for selecting genre, actor, random or quit
ask user for entry
validate entry
if valid, call void menu with selection

e. the complexity is O(n)

built-in data types: string, vector, multi map

user defined datatype

1. CreditDataSet:
   A. struct. the constructor does the heavy lifting: open file, while loop to read line by line (ignore header, increment index, throw line into stringstream, get column by column (movieID, title) and push to vectors cast dissect the cast block in while loop (find "name" field, check if at least 1 cast, create cast list, push cast list, increment index), return data.
   B. vector of movieID
   C. vector of title
   D. vector of cast
2. MovieDataSet:
   A. struct. The constructor does the heavy lifting: open file, while loop to read line by line (if header, ignore it, increment index, grab budget, grab genres, dissect genre block in while loop, grab homepage, grab movieID, grab keywords, grab original language, grab temporary title, grab overview block in while loop, grab popularity, grab production companies, grab release date, grab revenue, grab runtime, grab spoken languages, grab status, grab title, grab vote average, push the lot to vectors) ,return data
   B. vector of movieID
   C. vector of genre
   D. vector of title
   E. vector of overview
   F. vector of runtime
   G. vector of vote average
3. Movie Class
   A. movie title
   B. vote average
   C. genre
   D. need method to get title
   E. need method to get average
   F. need method to get genre
4. ActorHashMap
   A. multimap (due to multiple entries with same key needed and quick searching based on key)
      I. note: multimaps use pointers for access rather than .methods
   B. key: actor name

Brandon Cruz-Youll
Trinity Godwin
Yoshimasa Iwano
Armando Partida
Victor Uong

C. value: movie object that actor is in
D. need add movie method with key and value
E. need get movies method based on key search
    I. only return up to 5 movies to not be excessive

5. MovieHashMap
A. multimap (due to multiple entries with same key needed and quick searching based on key)
    I. note: multimaps use pointers for acess rather than .methods
B. key: genre
C. value: movie object that is in genre
D. need add movie method with key and value
E. need get movies method based on key search
    I. only return up to 5 movies to not be excessive

Strategies adopted by the team.

As a team, we did a rough design doc in the beginning to get us started. Using that process, we divided up the work. We relied heavily upon Discord and several Zoom meetings to hash out everything. We used GitHub for version control. We working up until the deadline pretty much. We would review each other's code by pulling the latest version from the repo and looking at the code.

We were working up until the end. In a perfect world, we would have deleted the actorHashMap and just called it MovieMultiMap. We ended up using the MovieHashMap for both the actor and the genre and were toying with the idea of not using separate maps and just using one giant map. Since the way we'd use the map would not matter how big it is. We first implemented the maps and created them each time we went through and chose an option in the menu. But then, we later discussed creating those in main so we wouldn't be creating and going through the process each time a choice was made.

Brandon worked on creating the hash maps, the design doc, editing the video and posting it to YouTube. He also did some peer code review and made suggestions for changes. He also was the admin for the group: contacting Professor Yang as needed, issuing Zoom meeting links and such. Trinity created the Movie object class. Yoshimasa worked on parsing the original files down to the data we needed while Armando worked on putting those into movie objects and populating the hash maps. Armando and Victor worked on the main file. Victor also contributed to parsing the files as well as putting those into movie objects and populating the hash maps. Victor also participated in peer review and making suggestions for changes.