CPT-281 Team Project 1-A: Movie Management System

Contributors: Jordan Pham, Athul Jaishankar

Project Summary:

This project is a Movie Management System that helps in maintaining two lists of movies: movies currently "showing" in the theater and movies "coming" to the theater. The system utilizes doubly linked lists and iterators for efficient management of movie data.

Technical Requirements:

▪ A movie has the following attributes:

1) release_date (Date type).
2) name (string type).
3) description (string type).
4) receive_date (Date type).
5) status (enum type). This data field tells if a movie is received or released.

▪ Movies in the "showing" list have status of released; movies in the "coming" list have status of received.

▪ The file that keeps track of the movies is a plain text file. An original file input format is made based on this example:

```
Glass, 01/18/2019, Drama/Fantasy, 01/12/2019, released
Miss Bala, 02/01/2019, Mystery/Thriller, 01/17/2019, received
```

In the example above, each line stores a movie. The information of the movie is structured like this: name, release_date, description, receive_date, status.

Functionality:

- Display Movies: Display both "showing" and "coming" movies.
- Add Movie: Add a new movie to the "coming" list, considering various constraints.
- Start Showing Movies: Move movies from "coming" to "showing" list based on a specified release date, considering constraints.
- Edit Movie: Edit a movie in the "coming" list (e.g., update release date or description).
- Order Coming List: Keep the "coming" list ordered by release date.
- Count Coming Movies: Count the number of "coming" movies with a release date earlier than a specified date.
- Menu-based Interface: Implement a menu-based interface for user interaction.
- File I/O: Read movies from an initial file and write changes back to the file.
- This program uses the C++ built in STL for the list class.

System Design:

The Movie Management System was created and designed to effectively manage and organize movies in two lists: showing list and coming list. The System uses three main classes which are Date, Movie and Movie_Management_System.

- **Date class:**

  The Date class represents the date with day, month and year attributes. It is used to handle and validate date-related operations. One of the fundamental methods in the Date class would be “parse_from_string()” because it plays an important role in converting the string to a date object.

- **Movie class:**

  The Movie class represents a movie with attributes such as movie_name, release_date, description, receive_date and status. It includes getter and setter methods for adding, editing and displaying the film information.

- **Movie_Management_System class:**

  The Movie_Management_System class is the core class that manages the list of movies. It includes methods to display movies, add movies, start showing movies, edit release dates, edit movie descriptions, count movies before a specified date, save data to a file and load from a file. The Key feature in this class would be the sorting algorithm; we achieved sorting by the order of release date in O(n) time complexity.

Data Structures:

- **Linked List:**

  The system uses the linked list data structure to maintain two lists: the showing list and the coming list. Linked Lists are dynamically sized and allow efficient insertion and deletion of movie objects. In Movie_Management_System class, “start_showing_movie()” method uses a Linked List called “matching_movies” to keep track of all the movies that match the specified date in the coming list so that adding and removing movies from the showing and coming lists would be easier.

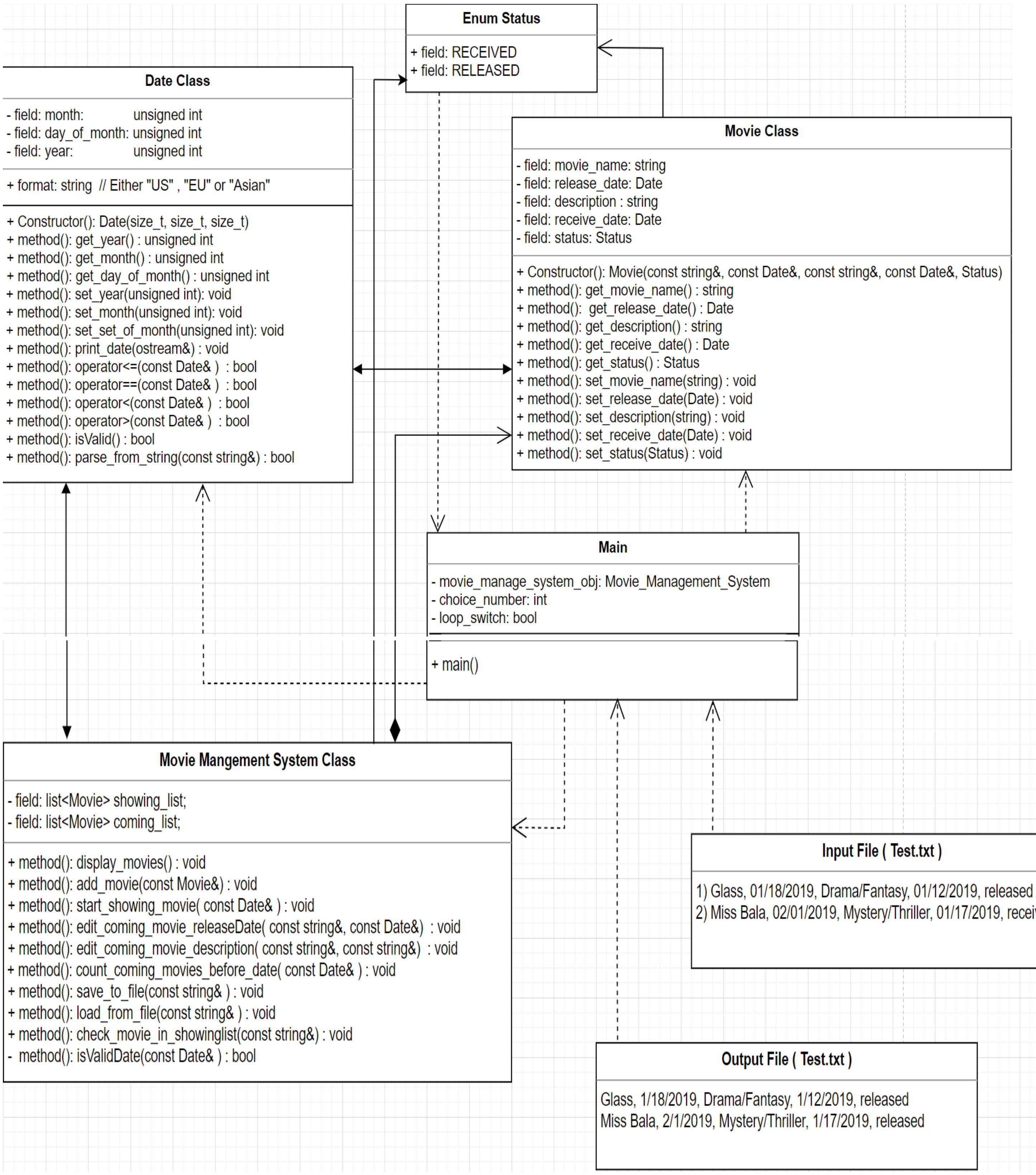
- **Enums:**

  The Status enum is used to represent the status of a movie, either “RECEIVED” or “RELEASED”.

- **String:**

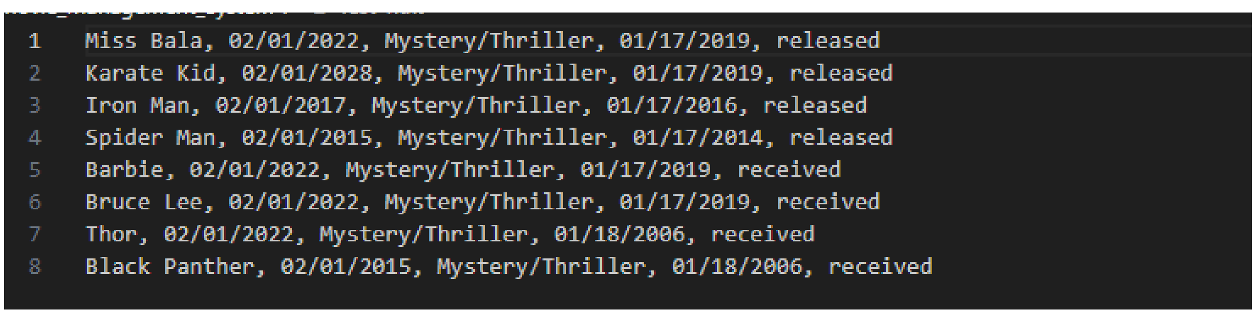
  String data type is used to store movie names, descriptions and formatted date strings.

UML:


Enum Status
+ field: RECEIVED
+ field: RELEASED
Date Class
- field: month: unsigned int
- field: day_of_month: unsigned int
- field: year: unsigned int
+ format: string // Either "US" , "EU" or "Asian"
+ Constructor(): Date(size_t, size_t, size_t)
+ method(): get_year() : unsigned int
+ method(): get_month() : unsigned int
+ method(): get_day_of_month() : unsigned int
+ method(): set_year(unsigned int): void
+ method(): set_month(unsigned int): void
+ method(): set_set_of_month(unsigned int): void
+ method(): print_date(ostream&) : void
+ method(): operator<=(const Date&) : bool
+ method(): operator==(const Date&) : bool
+ method(): operator<(const Date&) : bool
+ method(): operator>(const Date&) : bool
+ method(): isValid() : bool
+ method(): parse_from_string(const string&) : bool
Movie Class
- field: movie_name: string
- field: release_date: Date
- field: description : string
- field: receive_date: Date
- field: status: Status
+ Constructor(): Movie(const string&, const Date&, const string&, const Date&, Status)
+ method(): get_movie_name() : string
+ method(): get_release_date() : Date
+ method(): get_description() : string
+ method(): get_receive_date() : Date
+ method(): get_status() : Status
+ method(): set_movie_name(string) : void
+ method(): set_release_date(Date) : void
+ method(): set_description(string) : void
+ method(): set_receive_date(Date) : void
+ method(): set_status(Status) : void
Main
- movie_manage_system_obj: Movie_Management_System
- choice_number: int
- loop_switch: bool
+ main()
Movie Mangement System Class
- field: list<Movie> showing_list;
- field: list<Movie> coming_list;
+ method(): display_movies() : void
+ method(): add_movie(const Movie&) : void
+ method(): start_showing_movie(const Date&) : void
+ method(): edit_coming_movie_releaseDate(const string&, const Date&) : void
+ method(): edit_coming_movie_description(const string&, const string&) : void
+ method(): count_coming_movies_before_date(const Date&) : void
+ method(): save_to_file(const string&) : void
+ method(): load_from_file(const string&) : void
+ method(): check_movie_in_showinglist(const string&) : void
- method(): isValidDate(const Date&) : bool
Input File (Test.txt)
1) Glass, 01/18/2019, Drama/Fantasy, 01/12/2019, released
2) Miss Bala, 02/01/2019, Mystery/Thriller, 01/17/2019, recei
Output File (Test.txt)
Glass, 1/18/2019, Drama/Fantasy, 1/12/2019, released
Miss Bala, 2/1/2019, Mystery/Thriller, 1/17/2019, released

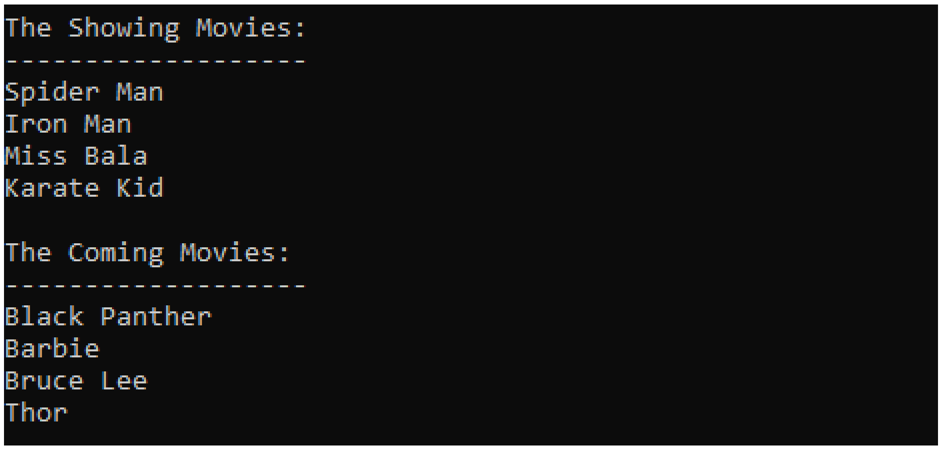
Test Cases

Test Case #1:
The first input file is shown below:


```
Miss Bala, 02/01/2022, Mystery/Thriller, 01/17/2019, released
Karate Kid, 02/01/2028, Mystery/Thriller, 01/17/2019, released
Iron Man, 02/01/2017, Mystery/Thriller, 01/17/2016, released
Spider Man, 02/01/2015, Mystery/Thriller, 01/17/2014, released
Barbie, 02/01/2022, Mystery/Thriller, 01/17/2019, received
Bruce Lee, 02/01/2022, Mystery/Thriller, 01/17/2019, received
Thor, 02/01/2022, Mystery/Thriller, 01/18/2006, received
Black Panther, 02/01/2015, Mystery/Thriller, 01/18/2006, received
```

The expected output would be to have movies separated into two categories: “The Showing Movies” and “The Coming Movies.” Movies in “The Showing Movies” are noted as “released” in the movie class and are sorted from earliest release date to latest release date. The movies in “The Coming Movies” are noted as “received” in the movie class and sorted from earliest date received to latest date received.

The output from the first test case is shown below:


```
The Showing Movies:
-------------------
Spider Man
Iron Man
Miss Bala
Karate Kid

The Coming Movies:
-------------------
Black Panther
Barbie
Bruce Lee
Thor
```

Test Case #2:

Team Member Contributions:

- **Athul Jaishankar – Team leader**

  - **Status.h:** Defined an enum called Status with two states: “RECEIVED” and “RELEASED”.
  - **Date.h:** Implemented a Date class with methods for checking validity, parsing from string and various getter and setter. Overloaded comparison operators to compare dates.
  - **Movie.h**: Defined a Movie class with attributes such as movie name, release date, description, receive date, and status. Implemented the getter and setter for each attribute.
  - **Movie_Mangement_System.h:** Implemented the Movie_Mangement_System class, which has methods for displaying movies, adding movies, starting to show movies, editing release date and descriptions, counting movies before a specified a date, saving to a file and loading from the file. Also implemented a helper function to check validity of a date.
  - **Main.cpp:** Created a main function to the Movie Management System application. Implemented a menu-driven interface to interact with the Movie Management System, allowing users to display movies, add movies, edit release dates and descriptions, start showing movies, count movies, save to a file and exit the program.
  - **Bug Fixes:** Addressed several issues mentioned in the comments, such as fixing the release date order, sorting movies by release date, and refining the logic for moving movies between the coming and showing lists.
  - **Project Management:** Scheduled and organized regular team meetings to facilitate communication and collaboration among team members. Designed the project outline, defining the structure and goals of the Movie Management System.
  - **Task Division:** Effectively divided tasks among team members, ensuring a balanced workload and efficient progress.

- **Jordan Pham – Team member**

  - **Documentation:** Started documenting the code with in-line comments, identifying completed tasks, bug fixes, and outlining remaining tasks.
  - **Header file Setup:** Set up each header file to ensure proper organization and manipulability, with a focus on the Date class.
  - **UML Diagram and Cover Page:** Created the UML diagram, providing a visual representation of the project's structure. Designed the cover page, ensuring a professional and cohesive project presentation.
  - **Team Meetings:** Attended team meetings, actively engaging in discussions and decision-making process.
  - **Test Case Creation:** Developed comprehensive test cases to ensure the robustness and reliability of the codebase.
  - **Debugging:** Played a key role in debugging the codebase, actively identifying and addressing the potential bugs and issues.
  - **Collaborative Work:** Worked collaboratively with team members, offering support and insights to enhance the overall quality of the project.

Future Improvements:

- **User Authentication:**

  Implementation of user authentication for secure access to the system. We can achieve this by creating usernames and passwords for logging into the system safely.

- **Search and Filtering:**

  Adding search and filtering options for efficient and fast retrieval of movie information. Designing an effective algorithm will be fundamental for performance.

- **GUI Interface:**

  Develop a graphical user interface for a more user-friendly experience.

- **Database Integration:**

  Integration of a back-end database system would be better for data management and persistence. By using open-source relational databases, long-term data could be archived and retrieved more efficiently.
- **Error Handling:**

  Enhancing error handling and providing meaningful error messages for better user guidance.
- **Sorting Algorithms:**

  Implementation of more efficient algorithms for large datasets would reduce the customer wait time.
- **Data Validation:**

  Improve input validation to handle various edge cases for detail.
- **Reporting:**

  Add reporting features to generate statistics and insights about movie data. We can perform graphical analysis for better comprehension.
- **Notification System:**

  Implementation of a notification system for important events or upcoming movie releases.
- **Creating rating and expiration date feature:**

  Create a rating and expiration date feature for gaining insights about customer satisfaction. Enabling customers to rate the movies would be useful for determining whether a movie should be displayed in the theater or not. Having an expiration date feature pushes the movie to the archive when the movie passes the expiration date. This would be helpful in keeping track of current movies and getting rid of unwanted movies.