National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Semester Project

Data Structures & Algorithms

Complex Engineering Problem

Section: BESE-11A **Session:** Fall-2021

Due Date: Friday, Dec 31, 2021

Instructions:

- One group shall have at max two members.
- Evaluation shall be carried out via viva exam and class presentation.
  a. Viva of the weaker student in a group shall determine the overall grade.
  b. Plagiarism cases will get zero marks.
- To get maximum marks:
  a. Assign meaningful names for variables, classes and functions.
  b. Use various data structures in this project like different implementations of lists.
  c. Use recursion where required in various operations like search, insertion, printing etc.
  d. Mention the worst-case and best-case complexities of each of the implemented functions.
- Use of array as data container is **discouraged** except in the cases where its requirement can be justified.
- Choose the most appropriate data structures, and searching and sorting algorithms where required.
- Name variables, classes and functions as per C++ programming conventions. Read the attached document entitled, “Naming Conventions in C++ - Coding Standards” prior to continue working on this project.
- Comment your code so that it is easy to understand.
  a. Above every function's code, a detailed description about its purpose and working principle should be provided.
- You must indent your code. Without it, your project won’t be evaluated.

Description:

Along, with this document you have been provided a .csv file containing a dataset of 5000 movies downloaded from the website of IMDB. For each of the 5000 movies, details of 28 parameters have been provided like actors, directors, rating etc. You should use this dataset to create a database using various data structures studied in the class.

One of your first tasks it to write a parser that extracts information from the csv file for each of the movies. To complete this project, you need to define the below mentioned classes, and implement all operations mentioned in **table 1**. Note that your approach to the problem in terms of defining classes, and functions and organizing data may be different than what is suggested in the guidelines below.

As discussed in the class, you shall need the following classes to organize data in various data structures in your application. Figures numbered 1 and 2 present a visual depiction of how you have to organize data. For details, watch the video recording already uploaded on LMS portal.

=======================**Movie** ==============================

A **Movie** Class that should:

a) Store data of all the 28 fields for a movie.
b) Remember, some of the fields like actors, plot keywords, genre etc. in the **Movie** class may contain more than one entry. Such entries should be stored in a suitable data structure such as an array-list or a linked list. You should justify the choice of your data structure during the viva/presentation exam. Similarly, plot keywords and genres of a movie are to be stored in a separate data structure. It is better to use enums.

============================ **MovieNode class** ===============

You have to stores all the movies given in the dataset in a suitable data structure. For doing so, you shall need to define a MovieNode class.

```
Class MovieNode{
        Movie data;
        MovieNode *next;
        MovieNode *prev; // in case you use a doubly linked list
}
```

You may declare an array of MovieNode type to store the dataset e.g. MovieNode MoviesArrayList[Size]. Make sure your decision is backed by logical reasons with focus on the reduction of time and space complexity.

================== **MovieCollection class** =======================

The MovieCollection class should provide the following functionalities:

a) A method that:
    a. reads the .csv file one line at a time.
    b. Creates an object of MovieNode class for each movie.
    c. Automatically inserts all the entries of that line in various fields of the data part in the MovieNode object. Note that the data field is of type Movie.
    d. Finally, insert the newly created MovieNode object in your list.
b) Once you have populated all your data structures, then you should implement the functionalities mentioned in table-1

================= **Director Node** =====================

DirectorNode class which should store

a) Director's name
b) A list of movies he has directed.
Hint: store a list of pointers to the objects of the movie nodes a director has directed.
c) Provide functionality to
    a. Search directors by name. The search should return name of directors along with the details of the movies they have directed.

Note: You shall store the data of all directors in a suitable data structure.

====================**Actor Node** ======================

ActorNode class which should store:

a) The Name of an actor
b) A list of the movies in which he/she has acted. Remember, movies' data is already stored in a list of type MovieNode class. Thus, the list of actor's movies should contain only pointers to the MovieNodes in which he/she has acted.
Note: Choose a suitable data structure.

======================================

A Visual Depiction of How Various Lists are to be Inter-Connected

As discussed in the class, the below given picture shows how various lists are to be inter-connected:

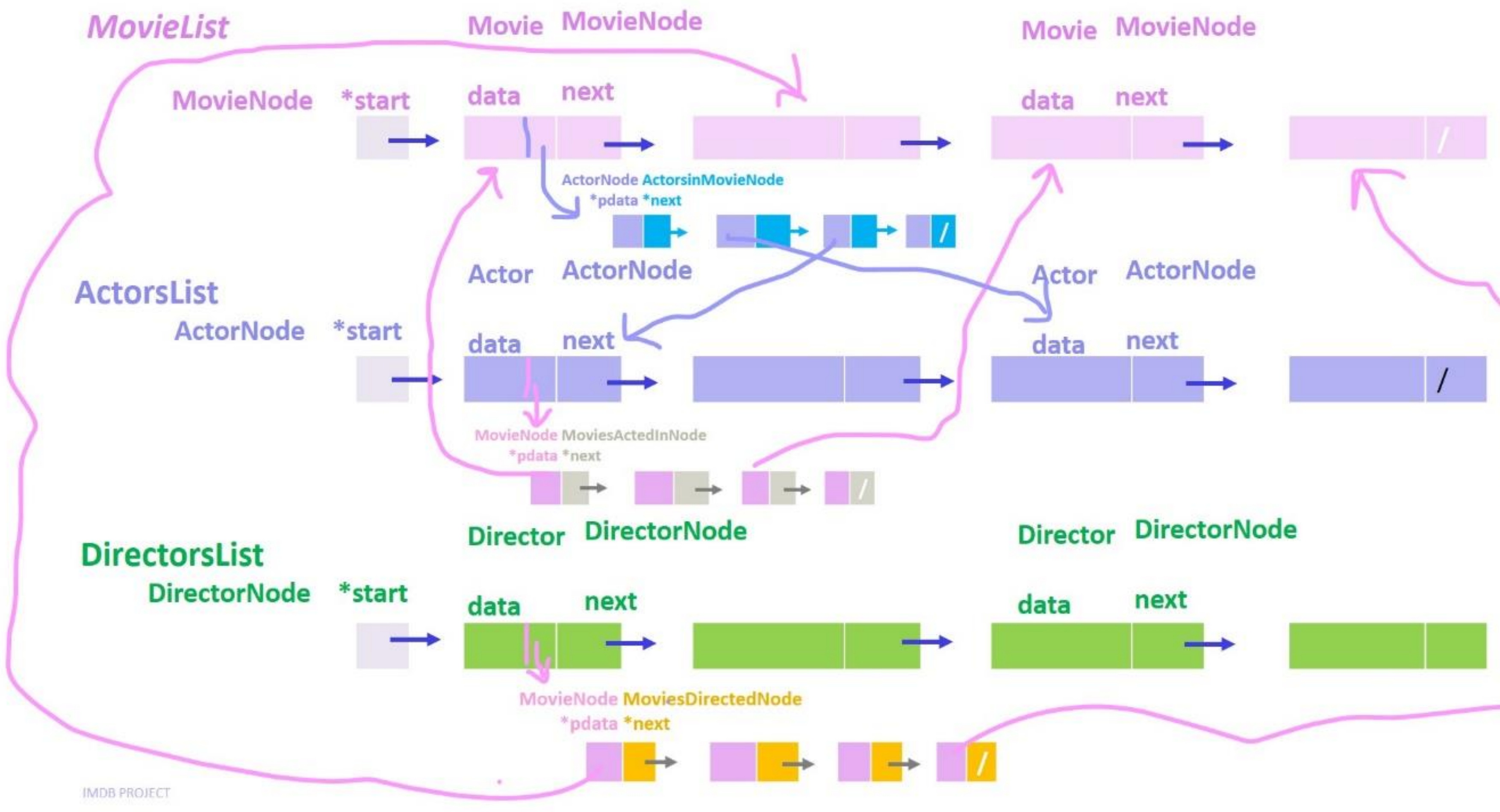

Figure 1: How to Store and Organize Data

Course: Data Structures and Algorithms

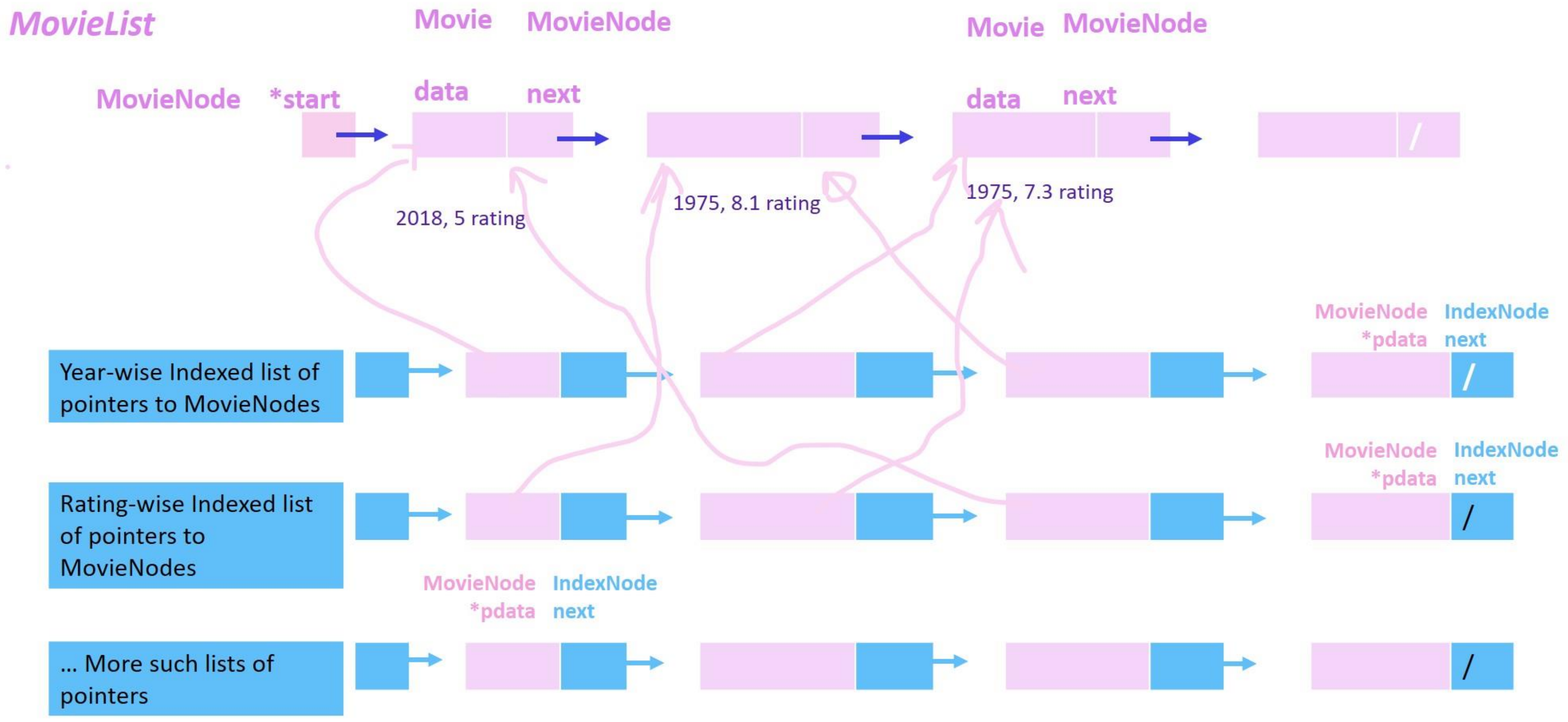

Figure 2: Indexed Lists of Pointers to Movie Nodes

Table 1: List of Functions to Be Implemented

#	Operation	Input	Output
Functions Related to Actors			
1	Search profile of an actor	Name of an actor	Actor's name, count of movies acted in, title of movies in chronological order and their year.
2	Search co-actors of an actor	Name of an actor	Print all co-actors of the given actor along with the title of the movies.
3	Search unique co-actors and print them.	Name of an actor	Print the name of all co-actors of an actor only once. For each co-actor, print title of the movies in which both acted in.
4	Print a list of all co-actors of the co-actors of an actor.	Name of an actor	Suppose an actor A has co-actors B and C in some movie. Each of B and C have their co-actors in other movies as well. For instance, B has co-actors D and E in another movie; C has co-actors F and G in a different movie. Given this example, this function shall print D, E, F and G. Note that these are co-actors of A's co-actors.
5	Checks whether A and B are co-actors or not?	Names of two actors e.g. A and B	If A and B have acted together in one or more movies, this function prints their titles. Else, it prints they have never acted together.
Functions Related to Directors			
6	Search director	Name of a director	Prints all movies a director has directed.
7	Print directors who have directed movies of a certain genre	Genre e.g. action	Prints the name of all those directors who have directed movies of the input genre type.
Functions Related to Movies			
8	Search a movie	Movie title (not necessarily complete)	Searches the movies list for the inserted title. If found prints information related to a movie such as its full title, release year, rating, duration, actors, director etc.
9	Search movies released in a given year	Release year	Prints all movie titles released in the given year. You may store a year-wise index of all movie titles separately.
10	Print movies year-wise	Options a) increasing order b) decreasing order	Prints movie titles year-wise. To print contents of a linked list in the reverse order, you may use a stack that stores pointers of movie nodes. Think about it!
11	Search movies based on genre	Genre e.g. action, adventure etc	Prints movies from highest to lowest rating. • You may maintain an indexed list of pointers to movie nodes for each genre. • Instead, your function may traverse the list of all 5000 movies and print only the titles for which the input genre matches. Whichever of these or another option you use, make sure to make decisions that improve the time and/or space complexity while considering nature of the fields in the IMDB dataset. Do provide the worst-case time complexity of your function.
12	Print movies rating-wise	NIL	Prints movies from highest to lowest rating. • You may maintain an indexed list of pointers to movie nodes sorted rating wise.

			• Or, your function may traverse the list of 5000 movies.
13	Print movies of a certain genre rating-wise	Genre	Prints all those movies rating-wise which match the input genre type.