

# CSCI-1200 Computer Science II — Spring 2006

## Homework 7 — Building a Movie Database

*Note: This is a two-week assignment that is worth twice as much as the one-week assignments we've had up to this point. You should get started on this assignment as early as possible.*

Many people now use the Internet Movie Database (IMDB, [imdb.com](http://imdb.com)) to answer their questions about movies, directors, and actors, and to find show times, reviews and trailers. Underlying IMDB is a fairly sophisticated software system. We are going to mimic part of that system in this assignment. Your job will be to parse this data, design appropriate data structures, and write functions to search and process the data in response to queries. *Please carefully read the entire assignment before beginning your implementation.*

### Command Line & Input/Output

The command line will include 3 input files containing people (actors & directors), movies, and queries. The queries will be requests for information about actors, directors, and movies. The output will be to `std::cout`. You may assume that all input is formatted properly, as described below. Example input and output is posted on the course webpage. Part of your assignment will be to make up new examples to demonstrate your implementation. The format of the data below is designed so you can easily cut & paste material from the [imdb.com](http://imdb.com) website. You are also welcome to invent new actors and movies as needed to thoroughly test your system.

### People Input

The first input file will contain a list of biographies for various actors and/or directors. The first line of data for a person will be their name. Many people will have exactly two names, but your program should work for people with arbitrary number of names. The second line is their date of birth formatted as day of month, month (spelled out), and year. Starting on the third line will be a paragraph giving their biography. One or more blank lines separate the different people and the end of the file will indicate the end of the list of people.

### Movie Input

The second input file will contain a list of movies. The first line of data for a movie contains the movie title, followed by the year of its release in parentheses, and then the director. Starting on the second line is a list of the cast. Each line starts with an actor's name. The name may or may not be followed by the string "... " and then the name of the character the actor played. The data for a movie will end with one or more blank lines or the end of the file (just as in the input for the personalities).

You may assume that the titles of the movies are unique — meaning that no remakes of old movies are considered, at least not under the same title. Note that not all actors and directors listed with a movie will be in the people input file. You must add these names to the list of all people, but of course you will not have a biography or a date of birth. You may assume that all titles and names are spelled correctly.

### Queries

Below are the queries your program must respond to. In the description of the format of the queries, the strings in lower case are the exact strings that will be input to indicate the query, while strings in *CAPS* are the variables. The output in response to each query should indicate what the query was and then provide the requested information. Please use the examples posted on the course webpage as a guide.

**bio PERSON** Outputs all of the available information for the person including their date of birth, their biography, the list of movies they appeared in as actors, and the list of all movies they have directed.

Each list of movies (as actor or director) should be sorted in reverse chronological order, with the most recent movie first. If an actor was in more than one movie in a given year, the movies within that year should be ordered alphabetically. If an actor is in more than one role in a movie, the order of the output of the roles should be alphabetical for that movie. If a person is not in the database, just output “No information for *PERSON*”.

**year** *YEAR* Outputs all of the people who were born in that year as well as all the movies that were released that year. The people should be listed with their date of birth and be listed ordered by date of birth. The movies should be listed alphabetically. If both lists are empty, output the message “Nothing happened in *YEAR*”.

**bacon\_number** *PERSON* Searches for the shortest path from the person to the actor Kevin Bacon through movies via acting or directing credits. The path is output as shown in the examples. At the end the number of links in the path, a.k.a. the *PERSON*'s Bacon number, is displayed. If there is no path the message “*PERSON* is not linked to Kevin Bacon” is printed. For more information on Bacon numbers see [http://en.wikipedia.org/wiki/Bacon\\_number](http://en.wikipedia.org/wiki/Bacon_number).

**search** *STRING* Output all movies that have the given *STRING* in their title and all the people that have *STRING* in their name. Order the list of movies by the year the movie was released and alphabetically within the year. Output the people in chronological order from earliest birthday to latest birthday. See the examples for specifics. If no movies or people match the *STRING*, output the message “*STRING* not found in database”.

For extra credit, extend the search query to also look for matches of the string in the actor and director biographies and in the names of the characters in movies, or make up interesting new queries. Include examples of their use in your new test case(s). Describe your extensions in your `README.txt` file.

## Additional Notes

In order to complete this assignment you may use any technique we have covered in lecture, in lab, or in the book chapters. Much of what we have covered will be useful in writing the program. Beyond the general rules of good program design and specific rules about how various constructs are to be used, there are no particular requirements on how you should design your solution. On the other hand, poorly-designed solutions will be penalized more heavily than in previous assignments.

A challenging aspect of this assignment is getting the input correct. Use the `std::getline` function and variations of the line breakup function seen in lecture as needed. Test the input code carefully, independent of the rest of the program. Construct useful and interesting new test cases and submit them along with your assignment.

## Documentation & Submission

Do think about performance as you choose your data structures (vectors, lists, maps, pairs, sets, etc.). It's up to you to prioritize the efficiency of the various operations. It's ok if not all of the operations are optimally efficient. In your `README.txt` file please describe and justify the data structure and algorithm design choices you have made and discuss how alternative representation and implementation decisions would change the performance.

Do all of your work in a new folder named `hw7` inside of your CSII homeworks directory. Please use the provided template `README.txt` file for any notes you want the grader to read. **You must do this assignment on your own, as described in “Academic Integrity for Homework” handout. If you did discuss the problem or errors messages, etc. with anyone, please list their names in your `README.txt` file.** When you are finished please zip up your folder exactly as instructed for the previous assignments and submit it through the course webpage.