# CS 211 PA #5: Graphs

In this assignment, we will build a graph from the IMDB actor database, which will allow us to determine relationships between actors (e.g. [the six degrees of Kevin Bacon](https://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon)). To begin, you must convert the IMDB data files into a graph in C++. For our purposes, we'll be using their [names file](https://datasets.imdbws.com/name.basics.tsv.gz) (note – this is a BIG file). To make debugging simpler, I've included reduced.tsv, a reduced version of the full IMDB list, as well as basic.tsv, which contains only 3 made up items that can get you started. Let's begin by discussing the file format:

## Data File Format

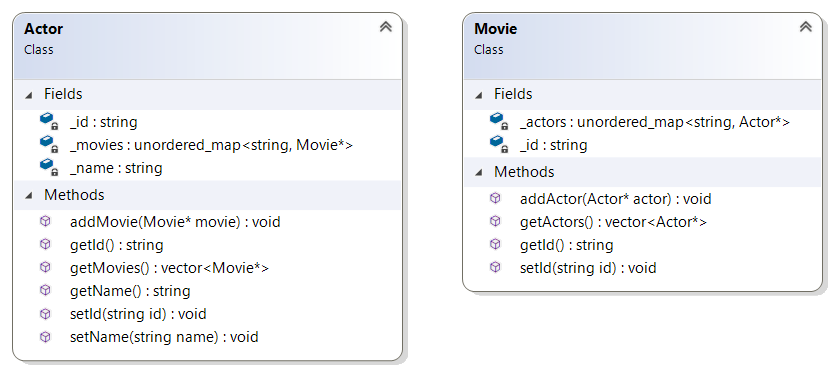
The IMDB data is in Tab Separated Value (TSV) format, which is very similar to CSV – instead of using commas to delineate cells, TSV uses tabs (\t) to delineate cells. Let's take a look at the first three lines of reduced.tsv:

|  |
| --- |
| nconst primaryName birthYear deathYear primaryProfession knownForTitles  nm0000001 Fred Astaire 1899 1987 soundtrack,actor,miscellaneous tt0072308,tt0043044,tt0050419,tt0053137  nm0000002 Lauren Bacall 1924 2014 actress,soundtrack tt0038355,tt0040506,tt0037382,tt0117057  nm0000003 Brigitte Bardot 1934 \N actress,soundtrack,producer tt0059956,tt0049189,tt0057345,tt0063715 |

The first line is a header row that defines the content of each column. For example, the first column is the actor's unique ID, the 2nd is the actor's name and the 3rd is when the actor was born. Following the header row are data rows. Because the data is too long to fit on a single line in this document, I've color coded each line so that you can more easily see where one line ends and another line begins. For the purposes of this assignment, we are most concerned with the actor's ID (col 0), name (col 1) and movie IDs that the actor is known for (col 5). Note that column 5 is in CSV format: Each movie ID is separated by a comma. It will be your task to parse a given text file into a graph using the class structure described in the next section.

## Graph Structure

Because our graph is comprised of both movies and actors, we will need to construct classes for both entity types. Included with this document are pre-made Actor and Movie classes that can be used in your assignment. Their structure is as follows:



## Tasks

In order to make this assignment both more approachable and easier to grade, I have broken down this assignment into a series of tasks. Note that examples are using the "reduced.tsv" file provided.

### Task #1: Parsing Data (EXTRA CREDIT – due 4/9)

The first step in doing something interesting requires us to convert from our text file into object form. Note that my StringSplitter.h file (included with this document) will be very helpful in completing this task. This task is extra credit and will be checked off during lab on April 9th. For those that do not complete the extra credit, I will provide an implementation for parsing TSV files on the same day.

### Task #2: Listing Movie Names (EXTRA CREDIT)

The names TSV file only contains movie IDs. To make these more meaningful, it would be helpful to display movie names in place of their IDs. In order to do so, you will need to pull data from [the IMDB movie names file](https://datasets.imdbws.com/title.basics.tsv.gz). Note that this file is also in TSV format. To complete this extra credit, you will need to replace movie IDs with their names in your program's output. Note that this might be easier if you modify the Movie class that I give you to include a title attribute.

### Task #3: Listing an Actor's Movies

In this task, you must list the movies associated with a specified actor. Example:

|  |
| --- |
| Enter Actor ID: nm0000010  Actor Name: James Cagney  Movie(s): tt0042041, tt0029870, tt0035575, tt0055256 |

### Task #4A: Connecting Actors

In this task, you must search the graph to determine whether or not two actors can be connected by one or more movies. Example:

|  |
| --- |
| Enter First Actor ID: nm0000012  Enter Second Actor ID: nm0000055  Actors are connected |

### Task #4B: Outputting Connected Path

Continuing from Task #4A, you must output the path of movies that connect the two actors using the shortest possible number of movie connections. Example:

|  |
| --- |
| Enter First Actor ID: nm0000012  Enter Second Actor ID: nm0000055  Actors are connected through movies:  \* tt0042192 |

### Task #5: Listing Associations by Degree

In this task, you will prompt the user for an actor and a degree number. Next, output all related actors and their distance in terms of the number of movies that separate the them. The algorithm for doing so is as follows:

1. Let ***degree*** = 1
2. Until ***degree*** is more than ***max\_degree*:**
3. For each movie associated with the primary actor, add other actors at ***degree*** level.
   1. For each actor in this list, increase ***degree*** by 1, do step #2

Example:

|  |
| --- |
| Enter Actor ID: nm0000059  Enter Max Degree: 3  Actor: Laurence Olivier  Degree 1 Connections: Joan Fontaine, Ursula Andress, Kirk Douglas  Degree 2 Connections: Sean Connery, Kirk Douglas  Degree 3 Connections: None |

## Starter Code

You are provided with foundational classes for Actor and Movie, which you are required to use. However, they may be modified to suit your needs. Also included are two data files (basic.tsv, reduced.tsv). The beginning of this document links to the full IMDB file which may be fun for testing. Lastly, I provide StringSplitter.h, which will greatly aid in your parsing of TSV files.

## Possible Strategy for Getting Started

1. Start with Task #3 as it is by far the easiest
2. Next, try for Task #4A as it's the most straight forward of the remaining tasks.
3. Depending on your implementation of Task #4A, completing Task #4B may be rather difficult. If this is the case, consider switching to Task #5 and looping back to 4B if you have the time.
4. Task #4B and Task #5 are related and have the potential to share a lot of code. It may be helpful to think about this before you tackle either.

## Header Comment, and Formatting

1. Be sure to modify the file header comment at the top of your script to indicate your name, student ID, completion time, and the names of any individuals that you collaborated with on the assignment.
2. Remember to follow the basic coding style guide. A basic list of rules is included with this document.

# Reflection Essay

In addition to the programming tasks listed above, your submission must include an essay that reflects on your experiences with this homework. This essay must be at least 350 words long. Note that the focus of this paper should be on your reflection, ***not*** on structure (e.g. introductory paragraph, conclusion, etc.). The essay is graded on content (i.e. it shows deep though) rather than syntax (e.g. spelling) and structure. Below are some prompts that can be used to get you thinking. Feel free to use these or to make up your own.

* Describe a particular struggle that you overcame when working on this programming assignment.
* Conversely, describe an issue with your assignment that you were unable to resolve.
* Provide advice to a future student on how he or she might succeed on this assignment.
* Describe the most fun aspect of the assignment.
* Describe the most challenging aspect of the assignment.
* Describe the most difficult aspect of the assignment to understand.
* Provide any suggestions for improving the assignment in the future.

## Deliverables

You must upload your program and reflection as a ZIP file through Canvas no later than midnight on Wednesday, April 25, 2018.

## PA #5 Checkins

During lab on 4/16, you must demonstrate a complete Task #3.

## Grading Criteria

Your assignment will be judged by the following criteria:

### Reflection essay (5pts)

* Your reflection meets the minimum requirements as specified earlier in this document.

### PA Checkin (5pts)

* You successfully demo your code during lab.

### Style (10pts)

* Your project contains good structure and implements the required classes. Your program intelligently uses classes when appropriate and generally conforms to good OOP design (i.e. everything isn't slapped into main).

### Dynamic Memory Management (5pts)

* You dynamically allocate memory in your program.
* You correctly clean up all dynamically allocated memory in your program.

### Task #1 (10pts EXTRA CREDIT)

* You successfully complete Task #1 before lab #11 on 5/9/2018

### Task #2 (5pts EXTRA CREDIT)

* You successfully complete Task #2 by incorporating movie titles into your searches.

### Task #3 (5pts)

* You successfully complete Task #3

### Task #4A (15pts)

* You successfully complete Task #4A

### Task #4B (25pts)

* You successfully complete Task #4B

### Task #5 (25pts)

* You successfully complete Task #5