ITP 365: Managing Data in C++

Homework 7: 6 Degrees of Separation Due: Friday, 12/1/2017 @ 11:59PM

Goal

In part A of this homework, you will implement a part of a program that will allow you to search through real data from IMDB. In part B, you will implement the "6 degrees of separation" for any arbitrary actor.

Lab Setup

- In this homework, there are five files:
  - main.cpp – Implements the main menu (do not change)
  - IMDBGraph.h/cpp – Implements the IMDBGraph and related structs such as ActorNode and Edge, which implement the Graph and related structs you will use (do not change)
  - The two files you may change are **IMDBData.h** and **IMDBData.cpp**.
- In IMDBData.h/cpp, make sure you update the file header comments so that your name and email is listed.
- For part B, you should only use the top250.list file for this homework – this file contains only the top 250 movies, according to votes on IMDB. The other file all.list contains every movie from IMDB and you should only use that for part A.
- This homework will NOT use the Stanford library, but instead only the STL library. Check the code I give you (or the interwebz) for the proper calling syntax.
- **Those of you using Visual Studio and Windows pretty much have to run part B in Release at all times – otherwise the program will run very slowly. Unfortunately, this means that debugging will be fairly limited for Windows users on this assignment.**

Part 1 (for part A): getMoviesFromActor/getActorsFromMovie

- Notice that in **IMDBData.h**, there are two member variables which are already declared for you:
  1. `mActorToMoviesMap` – This is a hash map where the key is a string that contains the name of an actor, and the value is a vector that contains all of the movies that actor has been in.
  2. `mMovieToActorsMap` – This is a hash map where the key is a string that contains the name of a movie, and the value is a vector that contains all of the actors that were in the movie.
- Neither of the above maps are being populated yet, but you will add this code in the subsequent parts of the lab.
- For now, you need to implement getMoviesFromActor and getActorsFromMovie:
  1. `getMoviesFromActor` – If the actor to movies map contains the actorName key specified, this function will return the lookup of the map. Make sure you use the [ ] to lookup, and not the get function. If the map does not contain the actor, you should return sEmptyVector (as the default return statement does).
  2. `getActorsFromMovie` – This function is essentially identical to the above function, but it instead looks into the movie to actors map.

- Just a note: there is no "containsKey" function in the `std::unordered_map` instead, you'll need to check to see if the find function of the hashmap finds the key like this:

  ```
  mActorsToMoviesMap.find(actorName) != mActorsToMoviesMap.end()
  ```

  This will be `true` if the `actorName` exists in the map.
- There really isn't a way to test these functions at the moment, but you will do so after the next part.

Part 2 (for part A): The Actor to Movies Map

- For this part, you will need to read in data from a .list file that contains actors. This data has been taken from IMDB (though I have simplified the format for you).
- Open up the `top250.list` file (on Windows, you should view it in Visual Studio or Notepad++ instead of just notepad). This file contains all of the actors and actresses who were in any movie in the "Top 250" on IMDB.
- You will notice that there first is a name of an actor, followed by one or more movies. Each movie is prefaced with the | symbol. For example:

  ```
  F. Murray Abraham
  |Amadeus (1984)
  |Scarface (1983)
  |The Grand Budapest Hotel (2014)
  ```

  …means that F. Murray Abraham was in three movies – *Amadeus (1984)*, *Scarface (1983)*, and *The Grand Budapest Hotel (2014)*.
- Every actor is guaranteed to have at least one movie, but as in the example above, an actor may have been in any number of movies.
- Your job is to implement the constructor of `IMDBData`. This constructor takes in the name of the file to load the data from. You must open the file using a `std::ifstream`, and then read in the lines. If you don't remember the syntax for opening and reading a file, you may want to consult your HW2 solution (when you loaded in the phone contacts file)
- When reading the file, **do not** remove the `(year)` part of the movie name.
- For each actor in the file, you should add an entry with the appropriate key (actor name) and value (vector of movies) into the `mActorToMoviesMap`
- To test out your code to this point, run the program and tell it to load `top250.list`. Then select option 1 and if you type in `Al Pacino`, the program should say:

  ```
  Al Pacino has been in 6 movies:
  Dog Day Afternoon (1975)
  Heat (1995)
  In the Name of the Father (1993)
  Scarface (1983)
  The Godfather (1972)
  The Godfather: Part II (1974)
  ```

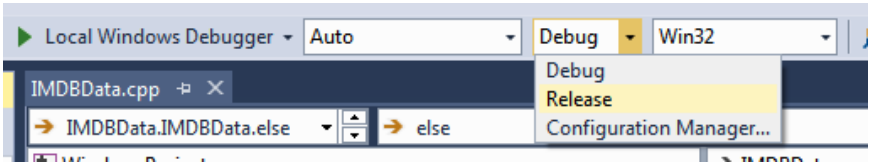
Part 3 (for part A): The Movie to Actors Map

- In this part, you will make the "reverse" map where every movie is associated with an actor. You will generate this using the actors to movie map you generated in Part 2. So you can't do this part until Part 2 works properly.
- To aid in this endeavor, you should first implement the `reverseMap` function. This function takes in the name of an actor as well as a vector which contains all of their movies. For each movie in the vector, you then want to do the following:
    1. If `mMovieToActorsMap` already contains the movie in question, you need to add `actorName` to that movie's associated vector
    2. Otherwise, you need to add an entry in `mMovieToActorsMap` that associates the movie with a vector that, for now, only contains `actorName`
- Once your `reverseMap` function is implemented, you then need to update the `IMDBData` constructor so that after it populates the `mActorToMoviesMap`, it then calls `reverseMap` on each actor to movie vector pair
- To test this part, once again load `top250.list`. This time, select option #2 and enter `Jurassic Park (1993)`. Though the order of the actors may vary, it should look something like:

```
Jurassic Park (1993) had 29 actors:
Dean Cundey
Laura Burnett
Laura Dern
Joseph Mazzello
Richard Attenborough
Ariana Richards
Jophery C. Brown
Lata Ryan
Martin Ferrero
Miguel Sandoval
Michael Lantieri
Brad M. Bucklin
Greg Burson
Brian Smrz
Tom Mishler
Whit Hertford
Adrian Escober
Christopher John Fields
Samuel L. Jackson
Jeff Goldblum
Richard Kiley
Wayne Knight
Gerald R. Molen
Sam Neill
Bob Peck
Gary Rodriguez
Cameron Thor
BD Wong
Robert 'Bobby Z' Zajonc
```

Testing on More Data (for part A)

- Once you've verified your code works on `top250.list`, you should then try the much bigger `all.list`
- Note that if you are on Windows, you will first want to change the build from "Debug" to "Release". A "Release" build is one where the code is more optimized. To change to "Release" click on the dropdown to the right of the play arrow that says "Debug" and change it to "Release":



  You also need to make sure you run with Ctrl+F5 instead of just F5
- On Mac, you don't need to do anything special – the default build in Xcode is fast enough
- In any event, once you are using the larger file you can test a much bigger set of actors and movies.
- For example, if you find out what movies `Kevin Bacon` has been in, you should get:

```
Kevin Bacon has been in 69 movies:
A Few Good Men (1992)
Animal House (1978)
Apollo 13 (1995)
...
Wild Things (1998)
X: First Class (2011)
```

  Alternatively, if you ask about what actors were in `Superman (1978)`, you should get:

```
Superman (1978) had 104 actors:
Gene Hackman
David Calder
Jill Ingham
...
David Yorston
Lawrence Trimble
Burnell Tucker
```

Part 4 (for part B): findRelationship – Basic BFS

- For this part, you will perform a basic BFS between two actors to see whether or not they have a connection. You'll find that most actors have a connection, but there are a few that do not.
- First, take a look at IMDBGraph.h. You'll see that the `IMDBGraph` class has a few different member functions that you will need to use:
    1. `containsActor`, which will tell you whether or not an actor is in the graph
    2. `getActorNode`, which returns a pointer to an `ActorNode`
- Note that the `ActorNode` struct has an adjacency list of `Edge` pointers, as well as a name for the actor and a visited bool.
- Finally, note that the `Edge` struct has the name of the movie (the label for the edge), as well as a pointer to the other actor
- For now, you will not need to use the `PathPair` struct (or the list of `PathPairs` in `ActorNode`)
- The code that is provided to you already creates an `IMDBGraph` for you (it's the `mGraph` member variable). Your job is to implement the BFS functionality inside `findRelationship`
- It is important you do not remove or move the `clearVisited` call at the bottom of the function. It must be there or your BFS will only work once per run of the program.
- For now, all you need to do is determine whether or not the two actors that are passed into `findRelationship` are connected. You do not need to actually specify what the path is.
- Before you begin the BFS, make sure you check that the two actors passed to `findRelationship` are in the graph – if they aren't you should cout an error message
- You'll want to implement BFS as was outlined in lecture. Keep in mind that the adjacency list in this case is a list of `Edge` pointers, not a list of `ActorNode` pointers. But you can find the connecting `ActorNode` by accessing the `mOtherActor` member of the Edge
- Don't forget that pointers use `->` to access members but references use `.` to access members
- Make sure you remember to set each `ActorNode`'s `mIsVisited` variable as you visit it, and you check it to make sure you don't revisit the same nodes (or your BFS will never finish)
- The BFS should finish in one of two cases:
    1. The queue becomes empty (in which case it failed to find a path)
    2. The second actor was found (it found a path)
- Once the BFS finishes, you should output an appropriate message such as "Found a path" or "Didn't find a path"
- The following pairs of actors should find paths when using the smaller file:
    - `Robert Redford` and `Robert De Niro`
    - `Paul Newman` and `Paul Bettany`
    - `Tatiana Zarubova` and `Masako Oshiro`
- The following pairs of actors should not find paths:
    - `Merila Zare'i` and `Orson Welles`

Part 5 (for part B): findRelationship – Showing the Path

- Once your basic BFS is working, it's time to display the paths between the actors
- To do this, you will need to use the `PathPair` struct as well as the list of `PathPairs` that the `ActorNode` struct has
- Each `PathPair` stores the name of a movie as well as an actor. Every time a new `ActorNode` is enqueued, you will keep track of the path from where you came from. This is how we will be able to reconstruct the sequence of movies and actors once a path is found. So you need to update each `ActorNode`'s `mPath` list while you are performing the BFS.
- To accomplish this, you need to add a bit more code. Whenever you enqueue an `ActorNode` into the queue, you need to also check to see if its `mPath` list is empty. If it is, you should set its `mPath` equal to the `mPath` of the current `ActorNode` you are visiting. You then need to append an additional `PathPair` to represent the additional "hop" of visiting.
- If the above doesn't make sense, be sure to consult the slides for the second graph lecture.
- Keep in mind that the `PathPair` constructor requires two parameters – the name of the movie and the name of the actor.
- You should then update your post-BFS output code to use a range-based for through the `mPath` list of the target `ActorNode`. You should be able to use this to output paths as in the sample output on the next page.

Sample Output

Here is some sample output once you've completed all parts of the homework (user input in red, some output is omitted to reduce the length). The exact path found may be different from the path below – but the number of hops should be the same:

```
Enter an actor file to load
>top250.list
...
>3
Enter the first actor's name
>Janet Zappala
Enter the second actor's name
>Laura Dern
--------------------------------------------------
Found a path!
Janet Zappala was in...
Twelve Monkeys (1995) with Bruce Willis who was in...
Pulp Fiction (1994) with Samuel L. Jackson who was in...
Jurassic Park (1993) with Laura Dern
--------------------------------------------------
...
>3
Enter the first actor's name
>Gail Yudain
Enter the second actor's name
>Su-kyeong Yun
--------------------------------------------------
Found a path! (4 hops)
Gail Yudain was in...
```

```
The Departed (2006) with Jeffrey Corazzini who was in...
Spotlight (2015/I) with Michael Keaton who was in...
Toy Story 3 (2010) with Sherry Lynn who was in...
Oldeuboi (2003) with Su-kyeong Yun
-------------------------------------------------
...
>3
Enter the first actor's name
>Sara Zamani
Enter the second actor's name
>Sean Young
-------------------------------------------------
Didn't find a path :(.
-------------------------------------------------
...
```

Submission

1. You must submit a ZIP file to Blackboard called **HW7Submit.zip**, using the designated Homework 7 submission link in the "Assignments" section.
2. Make sure to double-check that your zip file was created properly prior to submission. It is 100% your responsibility to check your submission.

Grading

Item	Points
Part 1 (for part A): getMoviesFromActor	5
Part 1 (for part A): getActorsFromMovie	5
Part 2 (for part A): The Actor to Movies Map	20
Part 3 (for part A): The Movie to Actors Map	15
Part 4 (for part B): findRelationship – Basic BFS	30
Part 5 (for part B): findRelationship – Showing the Path	15
Output text reasonably close to sample	5
Comments, style, and correct use of separate files	5
Total	**100**