CS 201, Fall 2019
Homework Assignment 1

Due: 23:55, November 15 (Friday), 2019

In this homework, you will implement a film collection system to store the lead actor names of films in a particular collection. For each film, you will store its title, its director, its year, its duration and its list of lead actors. In your implementation, the collection of films and the list of lead actors in each film will need to be implemented using **dynamically allocated arrays**. The homework has two parts whose requirements are explained below.

PART A: (30 points)

To take the final exam, you must submit this part and receive at least half of its points.

This part is a simplified version of the entire system. It stores the titles, directors, years and durations of films in a collection without their list of lead actors. The films will be stored in a dynamically allocated array of `Film` objects. Thus, you will first implement the `Film` class. This class is rather simple for Part A, but will need to be extended for Part B.

1. Below is the required part of the `Film` class. The name of the class should be `Film`; the interface for the class:

```
#ifndef __SIMPLE_FILM_H
#define __SIMPLE_FILM_H

#include <string>
using namespace std;
#include <iostream>

class Film{
public:
   Film(const string fTitle = "", const string fDirector = "",
        const unsigned int fYear = 0,
        const unsigned int fDuration = 0);
   Film(const Film &fToCopy);
   ~Film();
   void operator=(const Film &right);
   string getTitle() const;
   string getDirector() const;
   unsigned int getYear() const;
   unsigned int getDuration() const;

   friend ostream& operator<<(ostream& out, const Film& f);

private:
   string title;
   string director;
   unsigned int year;
   unsigned int duration;
};
#endif
```

should be coded in a file called `SimpleFilm.h` and its implementation should be coded in a file called `SimpleFilm.cpp`.

- The `Film` class should have the data members `title`, `director`, `year`, and `duration` (in minutes). You should implement get functions for these data members since they will be used to test your program.
- You should also implement a default constructor that initializes the `title`, `director`, `year`, and `duration` data members. Additionally, you should implement your own copy constructor and destructor, and overload the assignment operator and the << operator for outputting `Film` objects. Although you may use the default ones for some of these member functions, you are advised to implement them (although some may have no statements) in Part A so that it will be easier for you to extend them in Part B.
- Do not delete or modify any part of the given data members or member functions. However, you may define additional data members and member functions, if necessary.

2. Below is the required part of the film collection (`FC`) class that you will code in Part A of this assignment. The name of the class should be `FC`; the interface for the class:

```
#ifndef __SIMPLE_FC_H
#define __SIMPLE_FC_H

#include "SimpleFilm.h"

class FC{
public:
   FC();
   FC(const FC &fcToCopy);
   ~FC();
   void operator=(const FC &right);
   bool addFilm(const string fTitle, const string fDirector,
                const unsigned int fYear,
                const unsigned int fDuration);
   bool removeFilm(const string fTitle, const string fDirector);
   unsigned int getFilms(Film *&allFilms) const;

private:
   Film *films;
   unsigned int noOfFilms;
   unsigned int size;
};
#endif
```

should be coded in a file called `SimpleFC.h` and its implementation should be coded in a file called `SimpleFC.cpp`.

- Implement the default constructor, which creates an empty film collection. Also implement the copy constructor and destructor, and overload the assignment operator.
- Implement the add and remove film functions whose details are given below:

  Add a film: This function adds a film to the collection. The title, the director, the year, and the duration are specified as parameters. In this collection, the pair of title and director are unique (however, there can be multiple films with the same title and multiple films of the same director). Thus, if the user attempts to add a film with an already existing title and director, do not add the film and return `false`. Do not display any warning messages. Otherwise, if the film does not exist in the collection, add it to the collection and return `true`.

  Remove a film: This function removes a film from the collection. The title and the director are specified as parameters. If the given film exists in the collection, remove it from the collection

and return `true`. Otherwise, if there is no film with the given title and director, do not perform any action and return `false`. Likewise, do not display any warning messages.

- Implement the get function for films. The `getFilms` function should return the number of the films in the collection using the return value and the films in the collection by a pass-by-reference parameter called `allFilms`. Do not forget to put a deep copy of the `films` to the pass-by-reference parameter; otherwise, you may encounter run-time errors.
- Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

3. To test Part A, you should code your own main function in a separate file. Do not forget to test your code for different cases. However, do not submit any file containing the main function; otherwise, you may lose a considerable number of points. We will code our own driver to grade Part A of your assignment. In doing that, we will use the get functions of the `Film` and `FC` classes. Thus, do not forget to implement the get functions of these two classes. Below is an example of the test code and its output. In this example code, there is a global function called `displayAllFilms`. We will use it to display the films in the collection and to understand whether you add/remove films correctly. If you want, you may use this global function for your tests as well. Notice that the last line of this global function is to deallocate the `allFilms` array. Do not remove this line if you get any run-time errors while using the function. If you have such run-time errors, most probably, you

```
#include "SimpleFilm.h"
#include "SimpleFC.h"

void displayAllFilms(FC &fc){
   Film *allFilms;
   unsigned int noOfFilms = fc.getFilms(allFilms);

   cout << "No of films: " << noOfFilms << endl;
   for (unsigned int i = 0; i < noOfFilms; i++) cout << allFilms[i];

   if (allFilms != NULL) delete [] allFilms;
}

int main(){
   FC fc;

   fc.addFilm("Midnight Cowboy", "John Schlesinger", 1969, 113);
   if (fc.addFilm("Annie Hall", "Woody Allen", 1977, 93))
      cout << "Successful insertion of Annie Hall, 1977, "
           << "Woody Allen, 93 min" << endl;
   else
      cout << "Unsuccessful insertion of Annie Hall, 1977, "
           << "Woody Allen, 93 min" << endl;
   fc.addFilm("Full Metal Jacket", "Stanley Kubrick", 1987, 116);
   fc.addFilm("Good Will Hunting", "Gus Van Sant", 1997, 126);
   fc.addFilm("Requiem for a Dream", "Darren Aronofsky", 2000, 101);
   fc.addFilm("The Diving Bell and the Butterfly", "Julian Schnabel",
              2007, 112);
   if (fc.removeFilm("Zelig", "Woody Allen"))
      cout << "Successful deletion of Zelig, Woody Allen" << endl;
   else
      cout << "Unsuccessful deletion of Zelig, Woody Allen" << endl;
   displayAllFilms(fc);

   return 0;
}
```

made an error in implementing either the `getFilms` function or one or more of the destructor, copy constructor, and overloaded assignment operator.

The output should be:

```
Successful insertion of Annie Hall, 1977, Woody Allen, 93 min
Unsuccessful deletion of Zelig, Woody Allen
No of films: 6
Midnight Cowboy, 1969, John Schlesinger, 113 min
Annie Hall, 1977, Woody Allen, 96 min
Full Metal Jacket, 1987, Stanley Kubrick, 116 min
Good Will Hunting, 1997, Gus Van Sant, 126 min
Requiem for a Dream, 2000, Darren Aronofsky, 101 min
The Diving Bell and the Butterfly, 2007, Julian Schnabel, 112 min
```

What to submit for Part A?

You should put your `SimpleFilm.h`, `SimpleFilm.cpp`, `SimpleFC.h`, and `SimpleFC.cpp` files into a folder and zip the folder. In this zip file, there should not be any file containing the `main` function. The name of this zip file needs to be:

`PartA_secX_Firstname_Lastname_StudentID.zip`

where `X` is your section number. Then you should follow the steps that are explained at the end of this document for the submission of Part A.

What to be careful about implementation and submission for Part A?

You need to read the “notes about implementation” and the “notes about submission” parts that are given at the end of this document.

PART B: (70 points)

Now, Part A is to be extended such that each film has a list of lead actors. The full functionality of this extended `FC` system is to be provided. In order for this to be done, the `Film` class needs to be extended such that it additionally keeps the list of lead actors contained. The lead actor list of a film must be kept in a dynamically allocated array of `Actor` objects. Note that the number of lead actors can be different from one film to another, but is normally fixed for a particular film. The details of the classes are given below.

1. The requirements of the `Actor` class are given below. The name of the class should be `Actor`; its interface:

```
#ifndef __ACTOR_H
#define __ACTOR_H

#include <string>
using namespace std;
#include <iostream>

class Actor{
public:
   Actor(const string aName = "", const string aBirthPlace = "",
         const unsigned int aBirthYear = 0);
   Actor(const Actor &actorToCopy);
   ~Actor();
   void operator=(const Actor &right);
   string getName() const;
   string getBirthPlace() const;
   unsigned int getBirthYear() const;

   friend ostream& operator<<(ostream& out, const Actor& a);

private:
   string name;
   string birthPlace;
   unsigned int birthYear;
};
#endif
```

should be coded in a file called `Actor.h` and its implementation should be coded in a file called `Actor.cpp`.

- The `Actor` class keeps the name of an actor, her/his birth place and her/his birth year. The name should be unique for each film. That is, a film cannot have multiple actors with the exact same name but different films may have an actor with the same name. You should implement get functions for the data members since they will be used to test your program.
- Implement your own default constructor, copy constructor, and destructor, and overload the assignment operator and the << operator for outputting `Actor` objects.
- Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

2. The requirements of the `Film` class are given below. The name of the class should be `Film`. This time, the interface for the class:

```
#ifndef __FILM_H
#define __FILM_H

#include "Actor.h"

class Film{
public:
   Film(const string fTitle = "", const string fDirector = "",
        const unsigned int fYear = 0,
        const unsigned int fDuration = 0);
   Film(const Film &fToCopy);
   ~Film();
   void operator=(const Film &right);
   string getFilmTitle() const;
   string getFilmDirector() const;
   unsigned int getFilmYear() const;
   unsigned int getFilmDuration() const;
   unsigned int calculateAverageActorAge() const;

   friend ostream& operator<<(ostream& out, const Film& f);

private:
   string title;
   string director;
   unsigned int year;
   unsigned int duration;
   Actor *actors;
   unsigned int noOfActors;
   unsigned int size;
};
#endif
```

should be coded in a file called `Film.h` and its implementation should be coded in a file called `Film.cpp`.

- The `Film` class is similar to the one given in Part A. However, now it should keep a dynamic array of `Actor` objects as well. Similar to Part A, you should implement your own default constructor, copy constructor, and destructor, and do not forget to overload the assignment operator and the `<<` operator for outputting `Film` objects. Also, you should make sure to implement get functions for the `title`, `director`, `year`, and `duration` data members since they will be used to test your program.
- Your class should have a public member function called `calculateAverageActorAge` which will be used for testing. This function should calculate the average age of the leading actors in the film upon which it is invoked and return it. Note that in implementing this function, you should consider the existence of films in which there are no actors.
- Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

3. Below is the required part of the `FC` class that you will code in Part B of this assignment. The name of the class should be `FC`; the interface for the class:

```
#ifndef __FC_H
#define __FC_H

#include "Film.h"

class FC{
public:
   FC();
   FC(const FC &fcToCopy);
   ~FC();
   void operator=(const FC &right);
   bool addFilm(const string fTitle, const string fDirector,
                const unsigned int fYear,
                const unsigned int fDuration);
   bool removeFilm(const string fTitle, const string fDirector);
   unsigned int getFilms(Film *&allFilms) const;
   bool addActor(const string fTitle, const string fDirector,
                 const string aName, const string aBirthPlace,
                 const unsigned int aBirthYear);
   bool removeActors(const string fTitle, const string fDirector);
   unsigned int calculateAverageDuration() const;

private:
   Film *films;
   unsigned int noOfFilms;
   unsigned int size;
};
#endif
```

should be coded in a file called `FC.h` and its implementation should be coded in a file called `FC.cpp`.

- The `FC` class is similar to the one given in Part A. However, now it should also keep the list of leading actors for each film. Similar to Part A, you should implement your own default constructor, copy constructor, and destructor, and overload the assignment operator. Also you should make sure to implement the `getFilms` function, as explained before.
- Implement the following functions that your extended system should support.

  Add a film: This function adds a film to the collection. The title, the director, the year, and the duration are specified as parameters. In this function, the lead actor list is not specified; if there are any, lead actor(s) will be added later. In the film collection, the pair of title and director are unique (however, there can be multiple films with the same title and multiple films of the same director). Thus, if the user attempts to add a film with an already existing title and director, do not add the film and return `false`. Do not display any warning messages. Otherwise, if the film does not exist in the collection, add it to the collection and return `true`. This function is similar to what you will have implemented in Part A. But, in Part B, you should also create an empty lead actor list for the film when you add it to the collection.

  Remove a film: This function removes a film from the collection. The title and the director are specified as parameters. If the given film exists in the collection, remove it from the collection and return `true`. Otherwise, if there is no film with the given title and director, do not perform any action and return `false`. Likewise, do not display any warning messages. Note that this function should also clear the lead actor list of the specified film. This function is similar to what

you will have implemented in Part A. But now, for Part B, you should also remove the lead actor list when you remove the film from the collection.

Add a lead actor to the film: This function adds an actor to the lead actor list of a film in the collection. The title and the director of the film together with the name, the birth place, and the birth year of the actor are specified as parameters. In this function, you should take care of the following issues:

- If the specified film does not exist in the collection, do not perform any action and return `false`. Do not display any warning messages.
- All actor names are unique in a film. Thus, if the user attempts to add an actor with an existing name for the specified film, do not perform any action and return `false`. Do not display any warning messages. (However, different films may have an actor with the same name.)
- Otherwise, add the actor to the lead actor list of the specified film and return `true`.

Remove lead actor list from the film: This function clears the lead actor list of a film. The title and the director are given as parameters. If there is no film with the specified title and director, or if there are no actors in the lead actor list, do not perform any action and return `false`. Do not display any warning messages. Otherwise, clear the lead actor list of the specified film and return `true`.

Calculate the average duration of films in the collection: This function calculates the average duration of all films in the collection and returns the result. If there are no films in the collection, this function should return 0.

- Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

4. To test Part B, you should code your own main function in a separate file. Do not forget to test your code for different cases. However, do not submit any file containing the main function; otherwise, you may lose a considerable number of points. We will code our own driver to grade Part B of your assignment. In doing that, we will use the get functions of the `Actor`, `Film`, and `FC` classes. Thus, do not forget to implement the get functions of these three classes.

   Below is an example of the test code and its output. In this example code, there are two global functions called `displayAllFilms` and `displayStatistics`. The former is the same as the one implemented for Part A. The purpose of the latter is to obtain some statistics about films. We will use these functions to understand whether you add/remove films and actor lists correctly. If you want, you may use these global functions for your tests as well. Again the last line is included to deallocate the `allFilms` array. Do not remove this line if you get any run-time errors while using these functions. If you have such run-time errors, most probably, you made an error in implementing either the `getFilms` function or one or more of the destructor, copy constructor, and overloaded assignment operator.

```
#include "Film.h"
#include "FC.h"

void displayAllFilms(FC &fc){
   Film *allFilms;
   unsigned int noOfFilms = fc.getFilms(allFilms);

   cout << "No of films: " << noOfFilms << endl;
   for (unsigned int i = 0; i < noOfFilms; i++) cout << allFilms[i];

   if (allFilms != NULL) delete [] allFilms;
}

void displayStatistics(FC &fc){
   Film *allFilms;
   unsigned int noOfFilms = fc.getFilms(allFilms);

   if (allFilms != NULL) {
      unsigned int count[11] = {0}, mins, age, totalAge,
                   noOfFilmsWithActors = noOfFilms;

      for (unsigned int i = 0; i < noOfFilms; i++) {
          mins = allFilms[i].getFilmDuration();
          if (mins < 100) count[mins/10]++;
          else count[10]++;
      }
      for (int i = 0; i < 10; i++)
          if (count[i] > 0)
             cout << "Number of films with duration in [ "
                  << i*10 << "," << (i+1)*10 << ") min: "
                  << count[i] << endl;
      if (count[10] > 0)
         cout << "Number of films with duration >= 100 min: "
              << count[10] << endl;
      cout << "Average film duration: "
           << fc.calculateAvgDuration(); << " min, " << endl;

      totalAge = 0;
      for (unsigned int i = 0; i < noOfFilms; i++) {
          age = allFilms[i].calculateAverageActorAge();
          if (age == 0) noOfFilmsWithActors--;
          else totalAge += age;
      }
      if (noOfFilmsWithActors > 0) {
         age = totalAge/noOfFilmsWithActors;
         cout << "Average actor age: " << age << endl;
      }

      delete [] allFilms;
   }
}
```

```
int main(){
   FC fc;

   fc.addFilm("Midnight Cowboy", "John Schlesinger", 1969, 113);
   fc.addActor("Midnight Cowboy", "John Schlesinger",
               "Jon Voight", "Yonkers, NY, USA", 1938);
   fc.addActor("Midnight Cowboy", "John Schlesinger",
               "Dustin Hoffman", "Los Angeles, CA, USA", 1937);

   fc.addFilm("Annie Hall", "Woody Allen", 1977, 96);
   fc.addActor("Annie Hall", "Woody Allen",
               "Woody Allen", "Brooklyn, NY, USA", 1935);
   fc.addActor("Annie Hall", "Woody Allen",
               "Diane Keaton", "Los Angeles, CA, USA", 1946);

   fc.addFilm("Full Metal Jacket", "Stanley Kubrick", 1987, 116);
   fc.addActor("Full Metal Jacket", "Stanley Kubrick",
               "Matthew Modine", "Loma Linda, CA, USA", 1959);

   fc.addFilm("Good Will Hunting", "Gus Van Sant", 1997, 126);
   fc.addActor("Good Will Hunting", "Gus Van Sant",
               "Matt Damon", "Cambridge, MA, USA", 1970);

   fc.addFilm("Requiem for a Dream", "Darren Aronofsky", 2000, 101);
   fc.addActor("Requiem for a Dream", "Darren Aronofsky",
               "Ellen Burstyn", "Detroit, MI, USA", 1932);
   fc.addActor("Requiem for a Dream", "Darren Aronofsky",
               "Jared Leto", "Bossier City, LA, USA", 1971);

   fc.addFilm("The Diving Bell and the Butterfly", "Julian Schnabel",
              2007, 112);
   fc.addActor("The Diving Bell and the Butterfly",
               "Julian Schnabel", "Mathieu Amalric",
               "Neuilly-sur-Seine, France", 1965);

   fc.removeFilm("Full Metal Jacket", "Stanley Kubrick");

   displayAllFilms(fc);
   displayStatistics(fc);

   return 0;
}
```

The output should be:

```
No of films: 5
Midnight Cowboy, John Schlesinger, 1969, 113
   Jon Voight, Yonkers, NY, USA, 1938
   Dustin Hoffman, Los Angeles, CA, USA, 1937
Annie Hall, Woody Allen, 1977, 96
   Woody Allen, Brooklyn, NY, USA, 1935
   Diane Keaton, Los Angeles, CA, USA, 1946
Good Will Hunting, Gus Van Sant, 1997, 126
   Matt Damon, Cambridge, MA, USA, 1970
Requiem for a Dream, Darren Aronofsky, 2000, 101
   Ellen Burstyn, Detroit, MI, USA, 1932
   Jared Leto, Bossier City, LA, USA, 1971
The Diving Bell and the Butterfly, Julian Schnabel, 2007, 112
   Mathieu Amalric, Neuilly-sur-Seine, France, 1965
Number of films with duration in [ 90,100) minutes: 1
Number of films with duration >= 100 minutes: 4
Average film duration: 109 min
Average actor age: 36
```

What to submit for Part B?

You should put your `Actor.h`, `Actor.cpp`, `Film.h`, `Film.cpp`, `FC.h`, and `FC.cpp` files into a folder and zip the folder. In this zip file, there should not be any file containing the `main` function. The name of this zip file should be `PartB_secX_Firstname_Lastname_StudentID.zip` where `X` is your section number. Then you should follow the steps explained at the end of this document for the submission of Part B.

NOTES ABOUT IMPLEMENTATION (for both Part A and Part B):

1. You must use dynamically allocated arrays. You will receive no points if you use automatically allocated arrays, linked-lists or any other data structures such as `vector/array` from the standard library.
2. You are not allowed to use any global variables or any global functions when implementing the classes. However, you may use global functions to test your program, but should not submit them.
3. Your code must not have any memory leaks. You will lose points if your code has memory leaks even though it produces correct output. To detect memory leaks, you may want to use Valgrind which is available at `http://valgrind.org`.

NOTES ABOUT SUBMISSION (for both Part A and Part B):

1. Conform to the rules given separately for Part A and Part B. That is, the name of classes, the name of `.h` and `.cpp` files, and the name of zip files should conform to the specifications given separately for Part A and Part B. Otherwise, you may lose a considerable number of points.
2. **You need to upload two zip files (one for Part A and the other for Part B) using the upload link on Moodle (all sections) by the deadline. Read "what to submit for Part A" and "what to submit for Part B" sections very carefully.**
3. No hardcopy submission is needed. The standard rules about late homework submissions apply. Late submissions need to be done by e-mailing your two zip files to your TA **Ahmet Furkan Yıldırım**.
4. Do not upload/e-mail any files containing the `main` function.
5. You are free to code your programs on Linux or Windows platforms. However, we will test your programs on "`dijkstra.ug.bcc.bilkent.edu.tr`" and we will expect your programs to compile and run on the `dijkstra` machine. If we cannot get your program to properly work on the `dijkstra` machine, you will end up losing a considerable number of points. Therefore, we recommend you to make sure that your program compiles and properly works on "`dijkstra.ug.bcc.bilkent.edu.tr`" before submitting your assignment.
6. This assignment will be graded by your TA **Ahmet Furkan Yıldırım** (**`furkan.yildirim at bilkent edu tr`**). Thus, you may ask your homework related questions directly to him.