CS 201 - Spring 2018

Homework Assignment 3

Due: 23:59, May 7, 2018

In this homework, you will implement an academic conference system (ACS). A conference has a a list of tracks. A track represents a specific research field under the theme of the conference. Each track is denoted with a unique name and a list of academic papers submitted for that track. Each paper is identified by a unique name and a list of authors who contributed to the paper. In your implementation, you **MUST** use linked lists. This homework will have two parts, whose requirements are explained below.

PART A:

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

This part is a simplified version of the entire system, where the conference administrator just creates the conference and enters the tracks only. So the system does not contain any papers (so no authors either). In this system, you must keep the tracks in a linked list of the `Track` objects. Thus, you must implement the `Track` class first. This class is quite simple for Part A, but you will have to extend it for Part B.

1. Below is the required part of the `Track` class. The name of the class must be `Track`. The interface for the class must be written in a file called `SimpleTrack.h` and its implementation must be written in a file called `SimpleTrack.cpp`.
   - The `Track` class keeps the name of a single track as data members. Make sure to implement the get function for these data member since we will use them to test your program.
   - Implement the default constructor, which initializes the `trackName` data member. Additionally, implement your own destructor and copy constructor, and overload the assignment operator. Although you may use the default ones for some of these special functions, you are advised to implement them (some may have no statements) so that it will be easier for you to extend them for Part B.
   - 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.

```
#ifndef __SIMPLE_TRACK_H
#define __SIMPLE_TRACK_H

#include <string>
using namespace std;

class Track {
public:
   Track(const string tname = "");
   ~Track ();
   Track (const Track &trackToCopy);
   void operator=(const Track &right);
   string getTrackName();

private:
   string trackName;
};
#endif
```

2. Below is the required part of the `ACS` class that you must write in Part A of this assignment. The name of the class must be `ACS`. The interface for the class must be written in a file called `SimpleACS.h` and its implementation must be written in a file called `SimpleACS.cpp`. Do not delete or modify any part of the

given data members or member functions. You are not allowed to define additional functions and data members to this class for Part A .

```
#ifndef __SIMPLE_ACS_H
#define __SIMPLE_ACS_H

#include <string>
using namespace std;
#include "SimpleTrack.h"

class ACS{
public:
    ACS ();
    ~ACS ();
    ACS (const ACS& acsToCopy);
    void operator=(const ACS& right);
    bool addTrack(const string trackName);
    bool removeTrack (const int trackName);
    void displayTracks();

private:
    struct TrackNode {
            Track t;
            TrackNode* next;
    };
    TrackNode *head;
    string name;
    string topic;
    int trackCount;

    TrackNode* findTrack(string trackName);

};
#endif
```

You must keep the tracks in a linked-list of `TrackNode`s whose head pointer is `TrackNode *head`. In this class definition, you also see the prototype of a private function called `findTrack`. You may want to implement such an auxiliary function and use it in your add and remove functions (then in some other functions for Part B). This function takes the name of a track, searches it in the linked list of tracks, and returns a pointer to the `TrackNode` that contains that track if the track exists in the system. Otherwise, it returns NULL. This auxiliary function may help you write more clear codes. However, if you do not want to use it, just define an empty function (with no statements) in your `SimpleACS.cpp` file.

Things to do:

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

  Add a track: This function adds a track to the system. The name of the track is specified as a parameter. In this system, track names are unique. Thus, if the user attempts to add a track with an already existing name, do not add the track and return false. Otherwise, if the track does not exist in the system, add the track to the system and return true. DO NOT display any warning messages. Note that names are case insensitive (i.e., Machine Learning and MACHINE LEARNING are the same thing).

Remove a track: This function removes a track from the system. The name of this track is specified as a parameter. If the track with the given name exists in the system, remove it from the system and return true. Otherwise, if there is no track with the given name, do not perform any action and return false. Likewise, DO NOT display any warning messages.

Display all tracks: This function should display the names of all tracks in the system one per line. If the there are no tracks in the system, display `--EMPTY--.`

```
Track name1,
Track name2,
. . .
```

3. To test Part A, write your own main function in a separate file. Do not forget to test your code for different cases such that the system is created and and the above-mentioned functions are employed. However, do not submit this file. If any of your submitted files contains the main function, you may lose a considerable amount of points.

What to submit for Part A?

You should put your `SimpleTrack.h`, `SimpleTrack.cpp`, `SimpleACS.h` and `SimpleACS.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: PartA_secX_Firstname_Lastname_StudentID.zip where X is your section number. Then follow the steps that will be 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 have to read "notes about implementation" and "notes about submission" parts that will be given at the end of this document.

PART B:

In this part, you will extend the simple academic conference system. For this, first, you are supposed to implement the `Author` and `Paper` classes whose interfaces are given below, respectively. 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.

```
#ifndef __AUTHOR_H
#define __AUTHOR_H
#include <string>
using namespace std;

class Author{
public:
    Author();
    Author(const int id, const string name);
    int getID();
    string getName();

private:
    string name;
    int id;
};
#endif
```

```
#ifndef __PAPER_H
#define __PAPER_H

#include <string>
using namespace std;
#include "Author.h"

class Paper{
public:
    Paper();
    Paper( const string name );
    ~Paper();
    Paper( const Paper& paperToCopy );
    void operator=( const Paper& right );
    string getName();
    bool addAuthor( const int id, const string name );
    bool removeAuthor ( const int id );
    void displayAuthors();

private:
    struct AuthorNode {
            Author a;
            AuthorNode* next;
    };
    AuthorNode *head;
    string name;

    AuthorNode* findAuthor(int id);

};
#endif
```

Put the `Paper` class in a file called `Paper.h` and it's implementation in `Paper.cpp`, and put the `Author` class in a file called `Author.h` and it's implementation in `Author.cpp`. Implement all the functions given in the header files above. You must keep the authors of a paper in a linked-list of `AuthorNodes` whose head pointer is `AuthorNode *head`. In this class definition of a `Paper`, you also see the prototype of a private function called `findAuthor`. You may want to implement such an auxiliary function and use it in your add and remove functions. This function takes the id of an author, searches it in the linked list of authors, and returns a pointer to the `AuthorNode` that contains that author if the author exists in the system. Otherwise, it returns NULL. This auxiliary function may help you write more clear codes. However, if you do not want to use it, just define an empty function (with no statements) just like the version for tracks.

Here is some information about the functions to be implemented in the `Paper` class:

Add an author: This function adds an author to the paper. The id and name of the author are specified as parameters. In this system, author ids are unique. Thus, if the user attempts to enter an author that exists for that paper, display a warning message and do not perform the requested action.

Remove an author: This function removes an author from the paper. The id of the author is specified as a parameter. If there is no author with the given id, display a warning message and do not perform the requested action.

Display authors: This function lists all authors already added to the paper. The output should be in the following format. If the there are no authors of the paper, display `--EMPTY--`.

```
Author ID1, Author name1
Author ID2, Author name2
. . .
```

Then, extend the `Track` class from Part A, such that now it keeps the papers of a single track. These papers must be kept in another **LINKED-LIST**. Note that the number of papers in a track is not known in advance. Here, do not forget to implement the constructor, destructor, and copy constructor of this `Track` class as well as do not forget to overload its assignment operator. Otherwise, you may encounter some unexpected run-time errors. This time, the interface of the `Track` class must be written in a file called `Track.h`, and its implementation must be written in a file called `Track.cpp`.

After extending the `Track` class, now work on the implementation of the following functionalities that your ACS system should support:

1. Add a track
2. Remove a track
3. Display all tracks
4. Add a paper to a track
5. Remove a paper from a track
6. Add an author to a paper in a track
7. Remove an author from a paper in a track
8. Show detailed information about a particular track
9. Find the track(s) and paper(s) associated with a specific author id

Add a track: This function adds a track to the system. The name of the track is specified as a parameter. In this function, the paper list is not specified; the paper(s) (and their author(s)) will be added later. In this system, track names are unique (case insensitive). Thus, if the user attempts to enter a track with an already registered name, display a warning message and do not perform the requested action. This function is very similar to what you will implement in Part A. But now, for Part B, you will need to create an empty paper list for the track when you add it to the system.

Remove a track: This function removes a track from the system. The name of this track is specified as a parameter. If there is no track with the given name, display a warning message and do not perform the requested action. Note that this function also clears the paper list of the specified track as well as author lists for the papers. This function is very similar to what you will implement in Part A. But now, for Part B, you will need to remove its paper list along with the author lists of the papers when you remove the track from the system.

Display all tracks: This function lists all tracks already registered in the system. The output should be in the following format. If the there are no tracks in the system, display `--EMPTY--`.

```
Track name1
Track name2
. . .
```

Add a paper to the track: This function adds a paper to the paper list of a track. The track name for which the paper is submitted to and the name of the paper are specified as parameters. Note that the author list is initially empty and authors will be added later. In this function, you should take care of the following issues:

- If the track with the specified name does not exist in the system, display a warning message and do not perform the requested action.
- All paper names are unique (case insensitive) within the same track. Thus, if the user attempts to add a paper with an existing name in the same track, display a warning message and do not

perform the requested action. However, different tracks can have papers with the same name. Note that a paper can be submitted to multiple tracks.

Remove a paper from the track: This function removes a paper from the paper list of a track. Note that this operation should also clear the author list of the paper. The track name for which the paper is submitted to and the name of the paper are given as parameters. If there is no track with the specified name or if the specified paper name is not in the paper list of the specified track, display a warning message and do not perform the requested action.

Add an author to a paper in a track: This function adds an author to a paper in a given track. The track name for which the paper is submitted to, and the name of the paper are specified as parameters along with the author name and author id. In this function, you should take care of the following issues:

- If the track with the specified name does not exist in the system or the given paper does not exist in that track, display a warning message and do not perform the requested action.
- All author ids are unique. Thus, if the user attempts to add an author with an existing id in the same author list, display a warning message and do not perform the requested action. However, different papers can have authors with the same id.

Remove an author from a paper in a track: This function removes an author from the author list of the given paper in the given track. The id of the author, the track name for which the paper is submitted to and the name of the paper are given as parameters. If there is no track with the specified name or if the specified paper name is not in the paper list of the specified track, display a warning message and do not perform the requested action.

Show detailed information about a particular track: This function displays all of the information about a track whose name is specified as a parameter. The output should be in the following format. If the track with the specified name does not exist in the system, display `--EMPTY--` after the first line. Authors of a paper will be listed following the paper name and will start after a tab.

```
Track name
Paper name
   Author ID1, Author name1
   Author ID2, Author name2
Paper name
   Author ID1, Author name1
   Author ID2, Author name2
   Author ID3, Author name3
```

Find the track(s) and paper(s) associated with a specific author: This function lists all the tracks whose paper lists contain the specified a paper which is authored by the given author ID. Multiple tracks can have multiple papers, which can have the same author id. The output should be in the following format. If the specified author does not participate in any paper, display `--EMPTY--` after the author ID and name. Found paper names follow the track name and start after a tab.

```
Author ID, Author name
Track name (for the 1st track)
   Paper name (for the 1st paper in 1st track)
   Paper name (for the 2nd paper in 1st track)
Track name (for the 2nd track)
   Paper name (for the 1st paper in 2nd track)
   Paper name (for the 2nd paper in 2nd track)
   Paper name (for the 3rd paper in 2nd track)
      . . .
```

Below is the required public part of the ACS class that you must write in Part B of this assignment. The name of the class must be ACS. The interface for the class must be written in a file called ACS.h and its implementation must be written in a file called ACS.cpp. Your class definition should contain the following member functions and the specified data members. However, this time, if necessary, you may also define additional public and private member functions and data members in your class. You can also define additional classes in your solution. On the other hand, you are not allowed to delete any of the given functions or modify the prototype of any of these given functions.

```
#ifndef __ACS_H
#define __ACS_H
#include <string>
using namespace std;
#include "Track.h"
class ACS {
public:
	ACS();
	~ACS();
	ACS( const ACS& systemToCopy );
	void operator=( const ACS& right );
	void addTrack( string trackName );
	void removeTrack( string trackName );
	void displayAllTracks();
	void addPaper( string trackName, string paperName );
	void removePaper( string trackName, string paperName );
	void addAuthor( string trackName, string paperName, int authorID, string
authorName );
	void removeAuthor(string trackName, string paperName, int authorID );
	void displayTrack( string trackName );
	void displayAuthor( int authorID );
 private:
	struct Node {
		Track t;
		Node* next;
	};
	Node *head;
	int trackCount;
	Node* findTrack(string trackName);
};
#endif
```

What to submit for Part B?

You should put your Paper.h, Paper.cpp, Author.h, Author.cpp. Track.h, Track.cpp, ACS.h, and ACS.cpp (and additional .h and .cpp files if you implement additional classes) 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 follow the steps that will be explained at the end of this document for the submission of Part B.

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

You have to read "notes about implementation" and "notes about submission" parts that will be given just below.

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

1. You MUST use **LINKED-LISTs** in your implementation. You will get no points if you use fixed-sized arrays, dynamic arrays or any other data structures such as `vector/array` from the standard library.
2. Do not delete or modify any part of the given data members or member functions for the given classes. You are not allowed to define additional functions and data members for classes in Part A, but you may do so for Part B, if necessary.
3. You ARE NOT ALLOWED to use any global variables or any global functions.
4. Your code must not have any memory leaks for Part B. You will lose points if you have memory leaks in your program even though the outputs of the operations are correct.
5. Your implementation should consider all names as case insensitive.

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 the classes, the name of the .h and .cpp files, and the name of the zip files should conform to the specifications separately given for Part A and Part B. Otherwise, you may lose a considerable amount of points.
2. Before 23:59 on May 7, you need to send an email with a subject line "**CS201-HW3**" to **ONUR KARAKASLAR** (onur.karakaslar at bilkent edu tr), by attaching two zip files (one for Part A and the other for Part B). Read "what to submit for Part A" and "what to submit for Part B" sections very carefully. You may ask your homework related questions directly to him, as he will grade this homework.
3. No hard copy submission is needed. The standard rules about late homework submissions apply.
4. Do not submit any files containing the main function. We will write our own main function to test your implementations.
5. You are free to write your programs in any environment (you may use either Linux or Windows). On the other hand, 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 could not get your program properly work on the `dijkstra` machine, you would lose a considerable amount 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.