In this homework, you will implement a tournament team registration system using **LINKED-LISTs**. There are going to be multiple teams registering for the tournament. For each team you are going to store a record. Each team has a name, color, foundation year and a roster. In your implementation, you MUST use **LINKED-LISTs**. This assignment will be graded by your TA **ALPER KAGAN KAYALI (kagan.kayali at bilkent edu tr).** Thus, you may ask your homework related questions directly to him.

PART A:

This part is a simplified version of the entire system, which keeps just the name, color and foundation year of each team without keeping its roster. Below is the header file of the `SimpleReg` class that you must write in Part A of this assignment. Name this file as `SimpleReg.h`. **DO NOT CHANGE ANYTHING IN THIS HEADER FILE.** You may add additional data members and member functions, if necessary.

```
#ifndef __SIMPLE_REG_H
#define __SIMPLE_REG_H
#include <string>
using namespace std;
#include "SimpleTeam.h"
class SimpleReg {
   public:
           SimpleReg();
           ~SimpleReg();
           void addTeam( const string tName, const string tColor, const int tyear );
           void removeTeam( const string teamName );
           void displayAllTeams() const;

     private:
                struct Node {
                       Team t;
                       Node* next;
                };
                Node *head;
                int teamCount;
                Node* findTeam(const string teamName) const;
     };

#endif
```

As seen in this class definition,

1. You must keep the teams in a linked-list `Nodes` whose head pointer is `Node *head`. Thus, before starting the implementation of the `SimpleReg` class, you must define a class called `Team`. This `Team` class is quite simple for Part A (but you will have to extend this `Team` class for Part B). It will keep the name and the color of a single team as data members, and most probably (of course it depends on your implementation) the set and get functions for these data members. The interface for this class must be written in a file called `SimpleTeam.h` and its implementation must be written in a file called `SimpleTeam.cpp`.
2. You have to write the implementation of the `SimpleReg` class in a file called `SimpleReg.cpp`. Implement the default constructor, which creates an empty team registration system.
3. Then, implement add, remove, and display functions as well as the destructor whose details are given below. All of these functions are defined as public. In this class definition, you also see the prototype of a private function 2

called findTeam. 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 team, searches it in the teams array, and returns a pointer to the node that contains that team if the team 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 SimpleReg.cpp file.

Add a team: This function adds a team to the system. The name of the team, its color and its foundation year are specified as parameters. In this system, team names are unique (case insensitive). Thus, if the user attempts to enter a team with an already registered name, display a warning message and do not perform the requested action. Teams can have the same color and foundation year. However, a team cannot be founded in the future or before 1800. If such an attempt is made, display a warning message and do not perform the requested action.

Remove a team: This function removes a team from the system. The name of this team is specified as a parameter. If there is no team with the given name, display a warning message and do not perform the requested action.

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

```
Team name, color (for the 1st team), year (for the 1st team)
Team name, color (for the 2nd team), year (for the 2nd team)
. . .
```

Destructor: You are going to implement the destructor for the class, which will deallocate the dynamic memory.

4. To test Part A, write your own main function in a separate file. 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 SimpleTeam.h, SimpleTeam.cpp, SimpleReg.h and SimpleReg.cpp 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:

Now extend Part A such that each team will have a roster and provide the full functionality of this tournament team registration system.

For that, first, extend the Team class such that now it keeps the players (roster) of a single team. These players must be kept in another **LINKED-LIST**. Note that the number of players in a team is not known in advance. Here, do not forget to implement the constructor, destructor, and copy constructor of this Team class as well as do not forget to overload its assignment operator. Otherwise, you may encounter some unexpected run-time errors. Also implement the subscript operattor to reach teams. This time, the interface of the Team class must be written in a file called Team.h, and its implementation must be written in a file called Team.cpp.

After extending the Team class, now work on the implementation of the following functionalities that your system should support.

1. Add a team
2. Remove a team
3. Display all registered teams
4. Add a player to the team
5. Remove a player from the team
6. Show detailed information about a particular team

7. Find the team(s) whose roster contains a specified player name

Add a team: This function adds a team to the system. The name of the team, its color and foundation year are specified as parameters. Teams can have the same color and foundation year. Again, a team cannot be founded in the future or before 1800. If such an attempt is made, display a warning message and do not perform the requested action. In this function, the player list is not specified; the player(s) will be added later. In this system, team names are unique (case insensitive). Thus, if the user attempts to enter a team 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 roster for the team when you add it to the system.)

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

Display all teams: This function lists all teams already registered in the system. The output should be in the following format. If the there are no teams in the system, display `--EMPTY--`. (This function is exactly the same with what you will implement in Part A.)

```
Team name, color (for the 1st team), team year (for the 1st team)
Team name, color (for the 2nd team), team year (for the 2nd team)
. . .
```

Add a player to the team: This function adds a player to the roster of a team. The team name for which the player is playing, the name of the player, and its position (e.g., guard, forward, center) are specified as parameters. In this function, you should take care of the following issues:

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

Remove a player from the team: This function removes a player from the roster of a team. The team name for which the player is playing and the name of the player are given as parameters. If there is no team with the specified name or if the specified player name is not in the roster of the specified team, display a warning message and do not perform the requested action.

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

```
Team, color, year
Player name, position (for the 1st player)
Player name, position (for the 2nd player)
. . .
```

Find the team(s) whose roster contains a specified player name: This function lists all the teams whose rosters contain the specified player name. As multiple teams can have players who have the same name, given a name you might have multiple teams associated with this player name. The output should be in the following format. If the specified player does not participate in any roster, display `--EMPTY--` after the player name.

```
Player name
Position, team name, team color (for the 1st team), team year (for the 1st team)
Position, team name, team color (for the 2nd team), team year (for the 2nd team)
```

. . .

Below is the required public part of the CompleteReg class that you must write in Part B of this assignment. The name of the class must be CompleteReg. The interface for the class must be written in a file called CompleteReg.h and its implementation must be written in a file called CompleteReg.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 __COMPLETE_REG__H
#define __COMPLETE_REG__H
#include <string>
using namespace std;
#include "Team.h"
class CompleteReg {
public:
        CompleteReg();
        ~CompleteReg ();
        CompleteReg ( const CompleteReg &systemToCopy );
        CompleteReg& operator=( const CompleteReg &right );
        Team& operator[]( const int index );
        void addTeam( const string tName, const string tColor, const int tyear );
        void removeTeam( string teamName );
        void displayAllTeams() const;
        void addPlayer( const string tName, const string pName, const string pPosition );
        void removePlayer( const string teamName, const string playerName );
        void displayTeam( const string teamName ) const;
        void displayPlayer( const string playerName ) const ;
private:
        struct Node {
               Team t;
               Node* next;
        };
        Node *head;
        int teamCount;
        Node* findTeam(string teamName) const;
};
```

<u>What to submit for Part B?</u>

You should put your Team.h, Team.cpp, CompleteReg.h, and CompleteReg.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.

<u>What to be careful about implementation and submission for Part B?</u>

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

<u>NOTES ABOUT IMPLEMENTATION (for both Part A and Part B):</u>

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. You ARE NOT ALLOWED to change anything in the SimpleReg class (for Part A). However, if necessary, you may define additional data members and member functions to the CompleteReg class (for Part B).
3. You ARE NOT ALLOWED to use any global variables or any global functions.