CS142/IT501 **Project 2** Spring 2016

Given: Tue Apr 5
Due: Fri Apr 22, 10 p.m.

This project is to create a program that will help the local youth soccer association manage registrations to their summer league. The program should maintain a file containing a list of all the players. Each player entry in this file should consist of four lines:

name
year of birth
category
registration status

The possible categories are U6, U8, U10, U12, U14 and U17. The registration status is either *paid* or *not paid*.

The program should be interactive and provide the following functions:

1. *Start a new season*. The user provides the current year and the program deletes all the existing players from the file. Before deleting, the program asks for a confirmation.

2. *Add a player*. The user provides the name, year of birth and registration status of a player and the program adds an entry to the file. The category is automatically computed.

3. *Look up a player*. The user provides a name and the program displays the entry for that player or a message saying that the player was not found.

4. *Edit a player's information*. After searching for a player, the user is given the option of changing that player's information (name, year of birth, registration status). If needed, the category is recomputed automatically.

5. *Generate a list of all the players in a given category*. The user provides the name of a category and a file name. The program then writes to that file a list of all the players in that category. Players are written to the file in alphabetical order and in the format described above.

Here are some additional details:

- *Categories*: Ux stands for *Under x*. For example, U6 is for players that are younger than 6 years old and U8 is for players that are 6 or 7. Players younger than 4 or older than 16 cannot play in this summer league. If the user attempts to add such a player, the program should not do it and instead print an explanation.

- *Player ages*: The age of a player is computed by subtracting the year of birth from the year that the user provided at the beginning of the season. For example, if the year of the current season is 2016 and a player was born in 2004, we consider that the player is 12 years old.

- *Error checking*: When the user is asked to enter a year, the program should check that the user enters an integer and nothing else. Otherwise, the program should ask for the year again. The program should also check that the file opens properly. If not, the program should print a message and quit. The program does not need to check that the file is properly formatted.

As was the case for the first project, the assignment policy available on the course web site applies to this project with one exception: if you do the project as a team, you are allowed to divide the work with your teammates. You should do the specification and design of the program as a team but a lot of the implementation work and documentation can be divided.

You can also have two teammates work together on the implementation of the same part of the program. If you do this, make sure that every team member gets a chance to write some of the code.

Note that even if you divide the work with your teammates, you should expect to have to help each other out. If you decide to work alone, I strongly suggest that you build the program very gradually. I also recommend that you read Chapter 9 of the notes, *Object-Oriented Design*, before beginning this project.

Hand in the following:

- *A program specification*. Make sure it is complete, precise and easy to understand. Don't forget to fully specify the user interaction and the file format.

- *A design document*. This too should be complete, precise and easy to understand. Make sure your program is highly modular. If possible, the user interaction, the storage of the player data and the details of individual player entries should be isolated in separate components. Your program should store the list of players in main memory. Choose an appropriate data structure. Document possible alternatives and explain the reasoning behind your choice.

- *A component diagram* that shows the main methods and data members of each component.

- *Interaction diagrams* that illustrate some typical scenarios.

- *All your code*. The program should be organized into multiple files in the standard way. Make sure you write standard C++ code so I can compile it with Code::Blocks and g++.

- *A status report*. It should state whether your program works and, in case it doesn't work perfectly or is not complete, explain exactly what doesn't work or remains to be done.