CS142 **Project 2** Spring 2025

Given: Fri, Apr 4
Due: Fri, Apr 25, 10 p.m.

Overview This project is to create a program that will help the local youth soccer association manage registrations to their summer league. The program allows the user to add players to the league, edit a player's information, print lists of players and search for players using a variety of criteria.

For each player, the program will record the player's name, year of birth, category and registration status. The possible categories are U6, U8, U10, U12, U14 and U17. The registration status is either *paid* or *not paid*.

Note that first and last names can consist of multiple words. In addition, two different players can share the same first name or the same last name but, to keep things simple in this version of the program, assume that no two players share the same full name.

Views The program provides the user with two views. In the *main view*, the user is presented with a menu of commands such as adding new players and searching for players. No player is displayed in this view.

The *search results view* is entered after a search. In this view, the user browses through the players produced by the search by using the commands *next* and *previous*. The players are shown one at a time and ordered alphabetically by last name and then first name. The *exit* command allows the user to return to the main view.

In the search results view, the program displays the rank of the current player

in the search results and the total number of players in the search results. For example, when viewing the 14th player after a search that returns 26 players, the program might display "14/26" or "14 out of 26".

Additional commands are available in both views. Details are given in the next two sections.

Main view commands

1. *Start a new season.* The user provides a year and the program deletes all the existing players. 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 that player to its list. The category is automatically computed. Should run in time logarithmic in the total number of players.

3. *Search for players.* The user provides any combination of last name, first name, keyword, year of birth, registration status, and category. The program searches for players that match all the information provided and then enters the search results view. If no matching players are found, a message is printed and the program remains in the main view.

   If a string is specified as keyword, a player will be a match if their first or last name *contains* that string.

   All the searches can run in time linear in the total number of players, *except* when a string is specified for the last name. In that case, the running time should be logarithmic in the total number of players plus linear in the number of players with the given last name.

4. *Print a list of players.* The program asks the user for the name of a file and writes all the players to that file. Players are written by category. Within each category, they are written in alphabetical order.

5. *Display statistics.* The total number of players, the number who have paid and the number who haven't paid, in total and for each category.

6. *Stop the program.* Any changes to the list of players are saved for the next session.

Search view commands

1. *Search for players.* Same as in the main view. If no matching players are found, a message is printed and the program returns to the main view. The program searches all the players, not just those being displayed in the search view.

2. *Move to the next or previous player.* As explained above, these work in alphabetical order. And they're circular: from the last player, moving to the next player moves to the first player; from the first player, moving to the previous player moves to the last player.

3. *Edit a player's information.* The user can change the name, year of birth and registration status of the player currently displayed. If needed, the category is recomputed automatically. Note that the player should remain in the search results, at the same spot, even if the search results are now out of order, and even if the player no longer matches the search criteria. Should run in logarithmic time.

4. *Print a list of players.* The program asks the user for the name of a file and writes players to that file. Only the players produced by the last search are written to the file.

5. *Exit the search results view.* Returns the program to the main view.

6. *Stop the program.* As in the main view, any changes to the list of players are saved for the next session.

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.

The age of a player is computed by subtracting the year of birth from the year of the current season. For example, if the year of the current season is 2025 and a player was born in 2013, then the player is considered to be 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. If not, the program should ask for the year again. The program should also check that any data files it needs open properly. If not, the program should print a message and quit. When the program asks the user for a command, it should check that the command is valid. If not, it should print a message.

User interaction Many details of how the program interacts with the user have not been specified. You can use the file viewer, text editor and phone book programs for inspiration, or you can do it differently. Either way, the goal should be for an intuitive, easy-to-use program.[1]

[1] User interaction is the main topic of CS459 Human-Computer Interaction.

Teamwork As was the case for the first project, the assignment policy available on the course website applies to this project with one exception: you are allowed to do the project as a team of up to three students.

If you choose to do the project as a team, you should do the specification and design of the program as a team but a lot of the implementation work 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 each team member gets a chance to write some of the code.

Note that even if you divide the implementation work with your teammates, you should expect to have to help each other out throughout the project.

Advice Whether you work as a team or on your own, I strongly suggest that you build the program very gradually. For example, in a first version of the program, you could assume that every player has only a last name, that all the players are in the same category and that there is no search command. Then gradually make the program more complete, one feature at a time.

You should review Section 7.3 of the notes, *Design and Implementation of the Phone Book*, before beginning this project.

Documentation Your project should include the following documentation:

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

- *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. (See Section 7.3 of the notes.)

- *Interaction diagrams* that illustrate some typical scenarios. (See Section 7.3 of the notes.)

Submission Submit your project on Moodle by the deadline (even if it's not complete). If you work as a team, submit only one copy of your project. Include the following in your submission:

- *Documentation*. All the documentation specified above.

- *All your code*. The program should be organized into multiple files in the standard way. Make sure you write standard C++ code so we can compile it without problems. Your code should be easy to understand. Pay particular attention to indentation and choice of names for variables, classes and functions.

- *Any necessary data files*. For example, if your program saves the year of the current season to a file between sessions, you need to submit that file.

- *A status report*. It should state whether your program works. In case it doesn't work perfectly or isn't complete, explain exactly what doesn't work or what remains to be done. If you work as a team, list the names of the team members at the top of the report.