Lab 7

Let's organize the soccer team and the social network

(3 classes, between October 23 and November 10)

Objectives:

- Produce code that is well-constructed, well-presented and that compiles without error.
- Cut an existing program into subprograms and multiple files.
- Manipulate vectors passed as parameters.
- Distinguish between the use of parameters passed by value or by reference.
- Produce automatic tests.

General instructions:

1. Read all questions before coding.
2. Prepare a test set before coding.
3. Adopt the "3-file" decomposition: a main source file (common to all) and for each group of subprograms a source file and a header file.
4. Perform automatic tests in the main program using the test set for all questions marked with the **[*]** symbol.

Work to hand in:

At the end of the 3rd class (no extra time), you will hand in the source (.cpp) and header (.h) files of the code (fully commented) you have produced. Your main must include the main program using the subprograms you have defined, as well as all the automatic tests you have carried out. Your work will be taken into account as bonus/malus counting towards your S-101 grade.

Available on Moodle:

- The lab description
- A source file called foot-corrige.cpp
- A help file for presenting tests.

Exercise 1: Let's organize the soccer team

In this exercise, we work on exercise 1 from Lab4, the correction for which is given in the file `soccerLab4.cpp`. You can copy and paste statements from this file to write the required subprograms.

The aim is to define subprograms to handle the various actions in this exercise, to test them with automatic tests wherever possible (questions marked with the symbol [*]) and to use them in the main program. The subprograms must not handle the user interaction part. For this reason, be sure to define subprograms that do not require user input (all such input will be made in the main program).

1. Create a new project containing 3 files: `soccer.cpp` (which will contain the main function) and `vectors.h` and `vectors.cpp` (intended to contain the vector manipulation subprograms you are asked to define).
2. Copy the `team` vector declaration from the `soccerLab4.cpp` file (line 11) into the `soccer.cpp` file.
3. Based on lines 27 to 30 of the `soccerLab4`.cpp file, define a `display` procedure to display the contents of a string vector.

   Use this procedure to display the contents of the `team` vector in your `soccer.cpp` program.

   Perform the tests for this question manually. Add your tests as comment at the end of the source file containing the subprogram definitions.
4. **[*]** Based on lines 52 to 54 of the `soccer.cpp` file, define a `normalization` procedure to change a person's name to uppercase.
5. **[*]** Based on lines 63 to 80, write a `search` function to find a person in a vector. What are its parameters?

   The function must return -1 if the person is not present, otherwise it returns the index of the element containing the person. Remember to use the `normalization` subprogram in the `search` subprogram.
6. **[*]** Using the `search` function, write a `replace` function to replace one person with another in a vector. What are its parameters?

   If the person is present, the subprogram returns true, else it returns false.

Exercise 2: Let's organize our social network

In this exercise, we work on Lab5, the correction of which is given to you. You can copy and paste statements from this file to write the required subprograms.

The aim is once again to define, to test and to use subprograms to handle the various actions in this exercise. As in the previous exercise, be sure to define subprograms that do not require user input.

Test each subprogram thoroughly.

1. Create a new project containing 3 files: `mainFacebook.cpp` (which will contain the main function) and `functionsFacebook.h` and `functionsFacebook.cpp` (intended to contain the vector manipulation subprograms you are asked to define).
2. Write an `initialization` subprogram to initialize a social network based on a constant `NBMEMBERS` giving the number of users.
3. Write a `display` subprogram to display the network it as you did in Lab5.
4. **[*]** Write an `addFriends` subprogram that adds to the network a friendship relationship between two users (who are already in the network, we don't need to check).
5. **[*]** Write a subprogram that returns the number of friends in common between two people.
6. **[*]** Add the following subprogram to your project. Prepare a test set. Test the program and make the necessary modifications. Repeat the modification-test loop as many times as necessary.

```
// the groupFriends function returns true if 3 people form a group of
friends where each is friends with the other two, and false otherwise.
bool groupFriends(const vect<vect<bool>> &network, int a, int b, int c)
{
      bool res = false;
      if (network[a][b] == network[b][a] ==true) res = true;
      if (network[a][c] == network[c][a] ==true) res = true;
      if (network[c][b] == network[b][c] ==true) res = true;
      return(res);
}
```

7. **[*]** We wish to store the name of each user. To do this, we use a new vector where we store in the element with index i the name of user i in the network. Update your main program (and any subprogram you deem useful) to be able to use the name of the individuals and not their index when displaying and adding friendship relationships. **Careful**: this modification concerns both the display and input of network members.
8. (optional) Write a function `BigGroupOfFriends` that takes as parameters a 2D vector of Booleans representing a social network and a vector of indices representing a set of network users. The function returns true if all users in the set are friends with all other users in the set, and false otherwise.