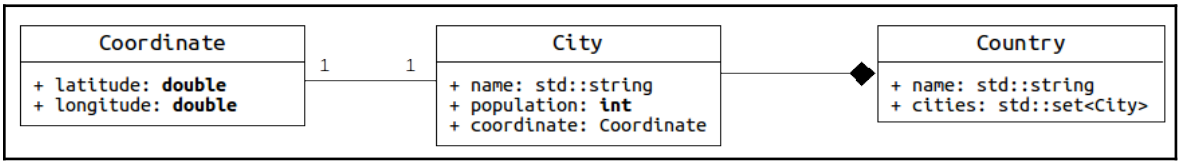
National Research University Higher School of Economics
Faculty of Computer Science
Bachelor's Program in Data Science and Business Analytics (DSBA)

Introduction to Programming

Part 1. Introduction to Structures in C++



(1) Declare structures `Coordinate`, `City`, and `Country` according to the UML *class diagram* shown above.

(2) Develop a function to fill a *container* of `Country` structures (passed by reference) with the file `cities.csv`.

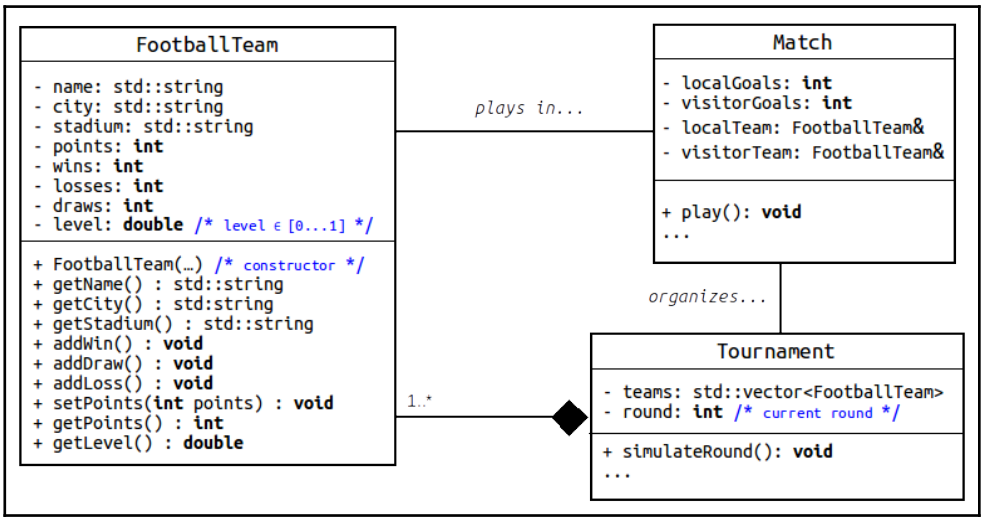
(3) Overload the operator << so that to print a `City` structure as follows:

```
(city = Shanghai, population = 22120000, coordinates = (31.16 , 121.46) )
```

Note: Consider the variant to overload the operator << for `std::set<City>` and/or for `Country`.

(4) Modify the ordering of cities inside the attribute `std::set<City> cities` of a `Country` structure, so that cities are internally ordered by their population ("*bigger cities go first*").

Part 2. Methods in Structures



(1) Implement a structure `FootballTeam` as designed in the UML class diagram shown above.

- Implement a constructor for `FootballTeam` that takes `name`, `city`, `stadium`, and `level` as arguments, whereas attributes `points`, `wins`, `draws` and `losses` are set to zero (0).

Note: Consider constructor variants which make use of default parameters and the constructor delegation feature.

(2) Implement the structure `Tournament` whose attribute `std::vector<FootballTeam> teams` is filled with the data from the file `football.csv`. The attribute `round` is initially set to zero (0) meaning the beginning of the tournament season.

(3) Implement the structure `Match` and its method `play()`.

- The method `play()` *simulates* a match between two football teams, a `localTeam` and a `visitorTeam`, and it updates the attributes of these two structures based on the winner of the match. See the algorithm for this method in the **Appendix 1**, *a*t the end of this file.
- Note: Attributes `localTeam` and `visitorTeam` of the Match structure are reference variables (&). T

(4) Implement the method `simulateRound()` of a Tournament structure.

- This method *simulates* a round of `N/2` matches between the `N` teams of the tournament.
- The `N/2` matches between teams as exemplified below: "*team k plays with team* `N-k+1`"

1	2	3	4	5	6	7
14	13	12	11	10	9	8

Example of a round with N = 14 teams.
Team #1 plays with team #14; team #2 plays with team #13, etc.

- After executing the `N/2` matches of a round, teams inside `std::vector<FootballTeam> teams` are re-ordered as exemplified below: Team #1 in the first position is always fixed, whereas the other teams inside the vector are *rotated clockwise one position*.

1	14	2	3	4	5	6
13	12	11	10	9	8	7

Re-ordering of teams in a vector teams after the round.

This rotation scheme allows that in the next round, different matches are organized, i.e., team #1 will play in the next round with team #13, team #14 will play with team #12, etc.

Additional exercises

(5) Overload the operator << for the structure `Match` so that to print the result of a match between two teams. For example, the `Match` structure may be printed as follows:

```
(localTeam=PFC Sochi, visitorTeam=CSKA Moscow, stadium=Fisht Olympic Stadium, result=0-3)
```

(6) At the end of the program, sort the vector `std::vector<FootballTeam> teams` by the attribute `point`. Then, print the vector to know the results of the tournament, i.e., the position of each team.

Appendix #1: *Simple match simulation algorithm* to implement the `play()` method of a `Match` structure.

Step 1. Calculate the number of goals in the match.

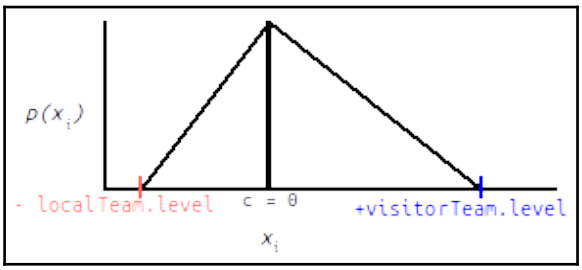
- Generate a random number *k* *uniformly distributed* in the range depicted below:

```
k ∈ [ 0,  ceil( abs(localTeam.level - visitorTeam.level) ) * 10 ]
```

"The bigger the difference between two teams, then bigger the chance to have more goals in a match"

Step 2. Distribute the number of goals between the teams.

- Generate *k* random numbers ("the goals") $\{X_1, X_2, \ldots, X_k\}$ using a *triangular distribution* in the range: `[-localTeam.level, +visitorTeam.level]` with center ("peak of the triangle") `c = 0`.



Example of a triangular probabilistic distribution in the range
`[-localTeam.level, +visitorTeam.level]` with `peak = 0.`
"The bigger of one of the two sizes of the triangle determines who scores more goals in average"

For each X_i ($1 \leq i \leq n$):

- if $X_i < 0$, then count a goal for the *local team i*n the match, i.e., `this->localGoals++`.
- Otherwise, then count a goal for the *visitor team* in the match, i.e., `this->visitorGoals++`.

Step 3. Decide the winner of the match to update the attributes of the `FootballTeam` structures. We shall assume that a *victory* gives 3 points to the winner, and 0 points to the loser. If both teams have a draw, i.e., then each team is granted with 1 point.

Thus, structures `localTeam` and `visitorTeam` are updated as follows:

Case 1:
`localGoals > visitorGoals`
"local wins, visitor losses"

```
localTeam.addWin();
localTeam.setPoints(3);

visitorTeam.addLoss();
```

Case 2:
`visitorGoals > localGoals`
"visitor wins, local losses"

```
visitorTeam.addWin();
visitorTeam.setPoints(3);

localTeam.addLoss();
```

Case 3:
`visitorGoals == localGoals`
"draw"

```
visitorTeam.addDraw();
visitorTeam.setPoints(1);

localTeam.addDraw();
LocalTeam.setPoints(1);
```