Course Id:	CMPE 260
Course Name:	PRINCIPLES OF PROGRAMMING LANGUAGES
Term:	SPRING 2018
Student Id:	2016400141
Student Name:	Halit Özsoy
Project Type:	Programming Project
Project Topic:	Programming in Prolog
Submission Date:	22nd April, 2018

CONTENTS

1. INTRODUCTION

This project includes Prolog predicates (statements) for manipulating lists of football teams, and the matches played between them.

The program uses 2 prolog files (as input), one for database, one for predicates.

Use your own database file consisting of two types of predicates as shown below.

1) `team(teamName, hometown).`
2) `match(week, homeTeam, homeTeamScore, awayTeam, awayTeamScore).`

Below is a sample database file, `cl_base.pl`:

```
team(realmadrid, madrid).
team(juventus, torinp).
team(galatasaray, istanbul).
team(kobenhavn, kopenag).

match(1, galatasaray, 1, realmadrid, 6).
match(1, kobenhavn, 1, juventus, 1).
match(2, juventus, 2, galatasaray, 2).
match(2, realmadrid, 4, kobenhavn, 0).
```

For the predicate file use the file, `predicates.pl`, shared in the appendix of this report.

After importing two such files into a prolog interpreter (`gprolog` or `swipl`), the following queries can be made by typing on standard input of the terminal, and the results are printed on to the standard output of the terminal.

- `allTeams(L, N).` or `allteams(L, N).`
- `wins(T,W,L,N).`
- `losses(T,W,L,N).`
- `draws(T,W,L,N).`
- `scored(T,W,S).`
- `conceded(T,W,C).`
- `average(T,W,A).`
- `order(L,W).`
- `topThree([T1,T2, T3],W).`

Examples:

-
  ```
  ?- allTeams(L, N).
  L = [realmadrid, juventus, galatasaray, kobenhavn, manutd, realsociedad,
  shaktard, bleverkusen, omarseille, arsenal, fcnapoli, bdortmund]
  N = 12
  ```

-
  ```
  ?- order(L, 6).
  L = [realmadrid, manutd, bdortmund, arsenal, fcnapoli, shaktard, juventus,
  bleverkusen, realsociedad, galatasaray, kobenhavn, omarseille]
  ```

-
  ```
  ?- draws(galatasaray,4,[juventus],1).
  True
  ```

Detailed info about those queries and how to run the program are given in the next sections of this report.

2. PROGRAM INTERFACE

2.1 HOW TO RUN

- Open a terminal in the same location (folder path) as the project files.
- Make sure the current path includes the files, "cl_base.pl" and "predicates.pl"
- Run the prolog interpreter by typing `gprolog` for Gnu Prolog or `swipl` for SWI-Prolog.
- Load the database by typing `[cl_base].` (Or any alternative database)
- Load the predicates by typing `[predicates].`
- Type any of the predicates (described below) to make queries.

Alternatively (on windows), run the gnu prolog application, then use `change_directory('path/to/prolog/project').` command. After that load the databases as described above. Instead of using bracket-syntax to load databases, the following commands can also be used: `consult('path/to/prolog/proeject/cl_base.pl').` (or any alternative database) and `consult('path/to/prolog/proeject/predicates.pl').` a

2.2 HOW TO TERMINATE

In order to terminate the program, either close the terminal or press `ctrl + d`. Alternatively, send interrupt signal by pressing `alt + ctrl + c` on Windows, or `ctrl + c` on Linux; then, press `e` to exit.

3. PROGRAM EXECUTION

3.1 GENERAL PROLOG QUERIES

- Type below commands on the terminal to make queries.
- Each query is of the form <predicate> (<arguments>)
- Each arguments may be a list, a string, a number or a variable.
- If an argument starts with an uppercase letter, than it's a variable. Otherwise, that argument is non-variable.
- For queries with at least one variable, each query is processed as the following: For which possible values for the given variables, is this predicate true when the non-variable arguments are as given. Return `false` if no such variables exist. If such variables can exists, returns a possible result. Then, returns next possible results after pressing `;` after each result. Returns `false` if no any other possible result exists. If enter is pressed instead of pressing `;` after a result, returns `true`, and doesn't show any more results for that query.
- For queries with all of its arguments are non-variable, returns `true` if such predicate is true with the given arguments, and returns `false` if such predicate is false with the given arguments.

3.2 ALL TEAMS

3.2.1 `allTeams(L, N)` or `allteams(L, N)`

- L is the list containing all the teams in the database where N is the number of elements in the list.
- Can specify queries with any combination of variables and constants (all permutations of the list of all teams, is a valid L) and the number of all teams is the only valid N.
- Type `allTeams(L, N)` to query all possible permutations of all the teams as L and the number of all teams as N. Type `;` after each possible result to see the next possible result. Press enter to stop.
- Type `allTeams(L, N)` but provide your own choice of L and N to see if they are valid configurations. (Returns `true`, if valid and `false` if not)
- Type `allTeams(L, N)` but provide your own choice of either L or N to see if there are any possible results for the other one. Returns `false` if there is not any possible result. Type `;` after each possible result to see the next possible result. Press enter to stop.

3.3 MATCH RESULTS

3.3.1 `wins(T, W, L, N)`

- Implies that L involves the teams defeated by team T when we are in week W and N is the number of elements in L.

3.3.2 `losses(T, W, L, N)`

- Implies that L involves the teams that defeated team T when we are in week W and N is the number of elements in L.

3.3.3 `draws(T, W, L, N)`

- Is very similar but now L involves the teams that team T could not defeat also did not lose to when we are in week W and N is the number of elements in L.

3.4 GOALS SCORED AND CONCEDED

3.4.1 `scored(T, W, S)`

- S is the total number of goals scored by team T up to (and including) week W.

3.4.2 `conceded(T, W, C)`

- C is the total number of goals conceded by team T up to week W. Assume T and W are given as constants.

3.4.3 `average(T, W, A)`

- A is the average (goals scored – goals conceded) of a team T gathered up to (and including) week W.
- It is assumed that T and W are given as constants. So, either use this query without any variables or with the only variable A.

3.5 ORDER AND TOP THREE

3.5.1 `order(L, W)`

- W (week) is given as constant and league order in that week will be retrieved in L.
- It is assumed that the order is decided according to averages on the specified week.
- i.e. The one with the highest average will be at the top.

- If the two teams have the same average then the order can be in any order.
- Returns false if no matches are played on the given week. (That week does not exists.)

3.5.2 `topThree([T1, T2, T3], W)`

- T1 T2 and T3 are the top teams when we are in the given week W.
- It is assumed that the order is decided according to averages on the specified week.
- i.e. The one with the highest average will be at the top.
- If the two teams have the same average then the order can be in any order.
- T3 is empty if there are less than 3 teams in total.
- T2 and T3 is empty if there are less than 2 teams in total.
- T1, T2, and T3 is empty if there are no teams.
- Returns false if no matches are played on the given week. (That week does not exists.)

4. INPUT AND OUTPUT

4.1 INPUT OUTPUT FORMAT

Run the program as below (type on terminal) to give inputs of database and predicates:

```
gprolog or swipl

[cl_base].

[predicates].
```

After that, use stdin and stdout to make queries and get responses.

4.2 INPUT (DATABASE)

A database file such as `cl_base.pl` containing lines of predicates as below:

```
team(teamName, hometown).

match(week, homeTeam, homeTeamScore, awayTeam, awayTeamScore).
```

4.3 INPUT (PREDICATES)

A file called `predicates.pl` created in this project. Can be found in the project zip and also in the appendix of this report.

4.4 INPUT (STDIN)

After executing, use the standard input (typing on the terminal window) to make queries for testing & usage of the program.

4.5 OUTPUT (STDOUT)

The result of each query is printed to the standard output by default as any other prolog program. Check out the terminal window for the results of each query.

5. PROGRAM STRUCTURE

5.1 ALL TEAMS

5.1.1 `allTeams(L, N)` or `allteams(L, N)`

- Use the built-in `findall` function as `findall(X, team(X, _), AllTeams),` to get all teams as a list `AllTeams`.
- Then, use the built-in `length` function to get the number of teams in that list as `N`.
- Then, use the built-in `permutation` function to accept any permutation of the list `AllTeams` as `L`.
- Make `allteams` an alias of `allTeams` by calling the other one.

5.2 MATCH RESULTS

5.2.1 `setWinLose(TeamScore, OtherScore, WinLose).`

- Set `WinLose` to 0 if Teams loses and Other wins.
- Set `WinLose` to 1 if Teams wins and Other loses.
- Set `WinLose` to 2 if Teams and Other draws.

5.2.2 `resultMatch(Team, Other, Week, WinLose).`

- Return a match result satisfying the given `WinLose` condition in the given `Week`.
- Call `setWinLose` function for each match to match the ones with the given `WinLose` condition.
- Defined twice, once to get matches where `Team` is at home, and once to get matches where `Other` is at home.

5.2.3 `results(T, W, L, ResultType).`

- Return all teams with whom the team `T` matched with a result satisfying the given `ResultType` for team, `T`; in and before the given week, `W`. (`ResultType` is the same as `WinLose` condition defined above. 0 for win, 1 for lose, and 2 for draw.)
- Use the built-in `findall` function with the `resultMatch` predicate to get all results of given type for each week.
- Use a recursive definition to get the results for the same function with week, `W` decremented by 1. (Return an empty list as the result if the week is 0, and stop the recursive iteration.)
- Append the results obtained from `findall` and the recursive call using the built-in function `append`, and return the result (merged list) of the `append` operation.

5.2.4 `resultsPermutation(T, W, L, N, ResultType).`

- Use `results` function above, to get the list of teams that team `T` matched with a result of `ResultType` in and before week `W`. Let the result of that function be `P`.
- Get the number of teams in that list by using the built-in `length` to get `N`.
- Set the list `L` to be any of the permutations of the list `P`, by using the built-in function `permutation`.

5.2.5 `wins(T, W, L, N)`

- Return a call to `resultPermutation` where the `ResultType` is 0 (win).

5.2.6 `losses(T, W, L, N)`

- Return a call to `resultPermutation` where the `ResultType` is 1 (lose).

5.2.7 `draws(T, W, L, N)`

- Return a call to `resultPermutation` where the `ResultType` is 0 (draw).

5.3 GOALS SCORED AND CONCEDED

5.3.1 `getScore(Team, TeamScore, Week).`

- Return a score for the `Team` scored on a match on `Week`.
- Two definitions, one for when the `Team` is at `Home`. And one for when the `Team` is `Away`.

5.3.2 `getConcede(Team, TeamConcede, Week).`

- Return a concede for the `Team` conceded on a match on `Week`.
- Two definitions, one for when the `Team` is at `Home`. And one for when the `Team` is `Away`.

5.3.3 `sumList([Head|Tail], Total).`

- Sum a given list `[Head|Tail]` into `Total`, using recursive definition and list concatenation.

5.3.4 `scoreWeek(T, W, S).`

- Returns the sum of scores that the team `T` scored on week `W`.
- Get all scores that team `T` scored on week `W`.
- Use built-in function `findall` with `getScore` to get all scores on list `Scores`.
- Sum these scores (the list `Score`) using `sumList` and return the result as `S`.

5.3.5 `concedeWeek(T, W, C).`

- Returns the sum of concedes that the team `T` conceded on week `W`.
- Get all concedes that team `T` conceded on week `W`.
- Use built-in function `findall` with `getConcedee` to get all concedes on list `Concedes`.
- Sum these concedes (the list `Concedes`) using `sumList` and return the result as `C`.

5.3.6 `scored(T, W, S)`

- Recursively get the sum of all scores that team `T` scored in and before `W`.
- Get the result of `scored` on week `W-1` `(Old)`, assign that result to `OldScore`.
- Get the result of `scoreWeek` for week `W` and assign that result to `CurrentScore`.
- Sum the `OldScore` and `CurentScore` to get the recursive sum of results. Assign that result to `S`.
- Return 0 on week 0, and end the recursive iteration.

5.3.7 `conceded(T, W, C)`

- Recursively get the sum of all concedes that team `T` conceded in and before `W`.
- Get the result of `conceded` on week `W-1` `(Old)`, assign that result to `OldConcede`.
- Get the result of `concedeWeek` for week `W` and assign that result to `CurrentConcede`.
- Sum the `OldConcede` and `CurentConcede` to get the recursive sum of results. Assign that result to `C`.
- Return 0 on week 0, and end the recursive iteration.

5.3.8 `average(T, W, A)`

- Get the score of the team `T`, in and before week `W`, using the function `scored` and assign the result into `Scores`.

- Get the concede of the team `T`, in and before week `W`, using the function `conceded` and assign the result into `Concedes`.
- Assign the difference of `Scores` and `Concedes` (`Scores - Concedes`) into the A, since that's the average of the team `T`, in and before week `W`.

5.4 ORDER AND TOP THREE

5.4.1 `isLower(Team, Week, BiggerTeams).`

- Select a team `Other`, that has a better average than `Team`, and not a member of the list `BiggerTeams`, on `Week`.
- Use built-in function `member` to check whether `Other` is a member of `BiggerTeams`.
- Returns `true`, if there's at least a team with higher average than `Team` that is not a member of `BiggerTeams`, on `Week`. Returns `false`, otherwise.
- When this function returns `false`, that means the given `Team` is the next best team (one with the next biggest average) after `BiggerTeams`.

5.4.2 `isBiggest(Team, Week, BiggerTeams).`

- Select team `Team`, that has a better average than any other team in `BiggerTeams`, on `Week`.
- Use built-in function `member` to check whether `Team` is a member of `BiggerTeams`.
- Use `isLower` function to check whether `Team` is the next biggest team after `BiggerTeams`.
- Successfully selects such team `Team` if there exists such team.

5.4.3 `getOrdered([Head|Tail], Week, N).`

- Recursively get `N` teams with highest average on week `Week`.
- The result, list, `[Head|Tail]` is reversely sorted. Meaning, the first element of it is the one with the highest average while the last element of it is the one with the lowest average.
- Call itself (`getOrdered`) with the `Tail` as the list for `N-1` as the `NextN`. Use list concatenation to combine the new element `Head` with the list `Tail`.
- Choose a `Head` by first using built-in `findall` function to get all possible next teams. (The one with the next highest average than `Tail`, but the result is a list (`BigTeams`) since it may contain more than one team with the same average.) Use built-in `member` function to choose one team as `Head` from the returned `BigTeams`.
- Recursive iteration ends when the `N` is 0, that the function returns an empty list.

5.4.4 `removeDuplicates([Head|Tail], [Head|Unique]).` `removeDuplicates([Head|Tail], Unique).`

- Recursively, removes duplicates in the list, `[Head|Tail]` and assigns the each unique element once to the new list, `[Head|Unique]` or `Unique`.
- Only removes duplicate and adjacent elements.
- If the first element of the `Tail` is equal to `Head`, than, the `Head` is repeated in `Tail`, so do not add it to the list `Unique`. (returns `Unique`)
- If the first element of the `Tail` is not equal to `Head`, than the `Head` is not repeated in `Tail`, so add it to the list `Unique` using list concatenation, now. (returns `[Head|Unique]`).

- Recursively call `removeDuplicates` for `Tail`, and determine the list `Unique` with this call.
- If the `[Head|Tail]` is an empty list, return an empty list as the `Unique`, and end the recursive iteration.

5.4.5 `getWeek(Week).`

- Get a week `Week` that in which at least match is played.
- Do not get the same week more than once.
- Use the built-in function `findall` with the database predicate, `match`, to get all weeks, assign that to `AllWeeks`.
- Sort the list `AllWeeks` into the list `SortedWeeks`, so that duplicate elements will be adjacent.
- Use the function `removeDuplicates` on `SortedWeeks` to get a list of all weeks that are not duplicated in that list, and assign the result into the list `UniqueWeeks`.
- Select a week `Week` from the list `UniqueWeeks`, using the built-in function `member`.

5.4.6 `order(L, W)`

- For a week `W`, return the ordered list of the teams on that week. (sorted from highest to lowest by the averages of all teams in and before week `W`)
- Select an existing week `W`, using the function `getWeek`.
- Find all teams using built-in function `findall` with database predicate `team` and assign the resulting list into `AllTeams`.
- Use the built-in `length` function on `AllTeams` to get the number of teams into `N`.
- Get the best `N` teams (all teams, since there are `N` teams in total) using the function `getOrdered` and assign its result into the `ReversedList`.
- Use the built-in function `reverse`, to reverse the `ReversedList` to obtain the result that goes from highest average to lowest average, assign that result to `L`.

5.4.7 `topThree([T1, T2, T3], W)`

- For a week `W`, return the ordered list of the top three teams on that week. (sorted from highest to lowest by the averages of all teams in and before week `W`)
- Returns all teams ordered from highest to lowest by their averages if there are less than three teams.
- Select an existing week `W`, using the function `getWeek`.
- Find all teams using built-in function `findall` with database predicate `team` and assign the resulting list into `AllTeams`.
- Use the built-in `length` function on `AllTeams` to get the number of teams into `MaxN`.
- Let `N` be the minimum of `3` and `MaxN` using the built-in function `min`. (in order to return all teams if `MaxN` is smaller than `3`)
- Get the best `N` teams (all teams, since there are `N` teams in total) using the function `getOrdered` and assign its result into the `ReversedList`.
- Use the built-in function `reverse`, to reverse the `ReversedList` to obtain the result that goes from highest average to lowest average, assign that result to `[T1, T2, T3]`.

6. EXAMPLES

6.1 All Teams

6.1.1 `allTeams(L, N)` or `allteams(L, N)`

```
?- allTeams(L, N).
L = [realmadrid, juventus, galatasaray, kobenhavn, manutd, realsociedad, shaktard,
bleverkusen, omarseille, arsenal, fcnapoli, bdortmund]
N = 12 ? ;
L = [realmadrid, juventus, galatasaray, kobenhavn, manutd, realsociedad, shaktard,
bleverkusen, omarseille, arsenal, bdortmund, fcnapoli]
N = 12 ? ;
L = [realmadrid, juventus, galatasaray, kobenhavn, manutd, realsociedad, shaktard,
bleverkusen, omarseille, fcnapoli, arsenal, bdortmund]
N = 12 ? ;
L = [realmadrid, juventus, galatasaray, kobenhavn, manutd, realsociedad, shaktard,
bleverkusen, omarseille, fcnapoli, bdortmund, arsenal]
N = 12 ?
(47 ms) yes

?- allteams([galatasaray, realmadrid, juventus, kobenhavn, manutd, realsociedad,
shaktard, bleverkusen, omarseille, arsenal, fcnapoli, bdortmund], 12).
True

?- allteams([], 12).
False

?- allteams(L, 12).
L = [galatasaray, realmadrid, juventus, kobenhavn, manutd, realsociedad, shaktard,
bleverkusen, omarseille, bdortmund, arsenal, fcnapoli]
True

?- allteams([galatasaray, realmadrid, juventus, kobenhavn, manutd, realsociedad,
shaktard, bleverkusen, omarseille, bdortmund, arsenal, fcnapoli], N).
N = 12
False
```

6.2 Match Results

6.2.1 `wins(T, W, L, N)`

```
?- wins(galatasaray,4,L,N).
L = [kobenhavn]
N = 1 ;
False
```

6.2.2 `losses(T, W, L, N)`

```
?- losses(galatasaray,4,L,N).
L = [realmadrid, kobenhavn]
N = 2 ;
False
```

6.2.3 `draws(T, W, L, N)`

```
?- draws(galatasaray,4,L,N).
L = [juventus]
N = 1 ;
False
```

```
?- draws(galatasaray,4,[juventus],N).
N = 1 ;
False

?- draws(galatasaray,4,L,1).
L = [juventus] ;
False

?- draws(galatasaray,4,[juventus],1).
True
```

6.3 Goals Scored and Conceded

6.3.1 `scored(T, W, S)`

```
?- scored(juventus,5,S).
S = 9
```

6.3.2 `conceded(T, W, C)`

```
?- conceded(juventus,5,C).
C = 8
```

6.3.3 `average(T, W, A)`

```
?- average(kobenhavn, 3, A).
A = -6

?- average(kobenhavn, 6, A).
A = -9
```

6.4 Order and Top Three

6.4.1 `order(L, W)`

```
?- order(L, 6).
L = [realmadrid, manutd, bdortmund, arsenal, fcnapoli, shaktard, juventus,
bleverkusen, realsociedad, galatasaray, kobenhavn, omarseille]

?- order([realmadrid, manutd, bdortmund, arsenal, fcnapoli, shaktard, juventus,
bleverkusen, realsociedad, galatasaray, kobenhavn, omarseille], W).
W = 6
```

6.4.2 `topThree([T1, T2, T3], W)`

```
?- topThree(L, 6).
L = [realmadrid, manutd, bdortmund]
```

7. IMPROVEMENTS AND EXTENSIONS

Since, that was my first prolog project, I am not knowledgeable about the programming conventions in the prolog. So, learning conventions and a rewrite of the whole code using those conventions can make the project more accessible, understandable or faster, etc.

8. DIFFICULTIES ENCOUNTERED

This program being my first prolog project, challenged me to learn and use prolog. Initially, I tried programming with a basic understanding of prolog. But then, I have gotten errors in my initial implementations that I had to trace for hours. But thanks to tracing, I've gotten a better understanding of how prolog works.

So, initially I've tried to program the allTeams function with just recursive functions and list concatenation. However, without knowing the limit (the total number of teams), my program would always go to infinite. After some debugging, I've found a way to list all possible combinations of teams with recursive functions. To do so, I had to change the order of conditions for predicates. By realizing this, I've understood that the conditions of predicates are processed one by one and in especially recursive situations, the order of those conditions make the predicates finite. I've managed to write an allTeams function with just recursive functions, that print all permutations of all the teams, but after the last permutation, it still went infinite.

Then, I've checked built-in functions, using the `findall` command, I could get the number of the all teams beforehand, and use it to limit my recursive functions so that they won't go infinite anymore. That indeed worked, but seeing also the `permutation` built-in command, I decided to use just this function instead of my 5-8 lines of recursive code. I thought, no need to reinvent the wheel. (After reinventing, though.) After that, I fully grasped the basics of prolog and had no more problems through the project. (Had to check the docs, of course, but that was to get reference (for example, to check built-in functions, operators, etc.) not because I had problems.

9. CONCLUSION

I've obtained a fine understanding of what prolog is, how it works, and how to write programs with it.

10. APPENDICES

10.1 PREDICATES.PL

```
/** Prolog Programming Project (Project 1)
CMPE 260 - Spring 2018, Boun
@author Halit Ozsoy - 2016400141
*/

% all teams in the database as a list (L) and its length (N)
% also defined as allteams, due to ambiguity on the project description and examples
allTeams(L, N) :-
    findall(X, team(X, _), AllTeams),
    length(AllTeams, N),
    % let N be the number of teams (length of AllTeams)
    permutation(AllTeams, L).
    % accept any answer L which is a permutation of AllTeams
allteams(L, N) :- allTeams(L, N).

% WinLose condition for the scores of two teams.
% 0 means Lose, 1 means Win, 2 means Draw for the first team. (Team)
setWinLose(TeamScore, OtherScore, WinLose):-
    TeamScore < OtherScore,
    WinLose is 0. % Team Loses
setWinLose(TeamScore, OtherScore, WinLose):-
    TeamScore > OtherScore,
    WinLose is 1. % Team Wins
setWinLose(TeamScore, OtherScore, WinLose):-
    TeamScore == OtherScore,
    WinLose is 2. % Draw

% WinLose condition of any two teams (Team, Other) in a week (Week)
resultMatch(Team, Other, Week, WinLose) :-
    match(Week, Team, TeamScore, Other, OtherScore),
    setWinLose(TeamScore, OtherScore, WinLose).
resultMatch(Team, Other, Week, WinLose) :-
    match(Week, Other, OtherScore, Team, TeamScore),
    setWinLose(TeamScore, OtherScore, WinLose).

% all teams that team T has won/lost/drawed (ResultType) against (L) in and before week W
results(_, 0, [], _).
results(T, W, L, ResultType):-
    W > 0,
    NextW is W - 1,
    results(T, NextW, OldResults, ResultType),
    findall(X, resultMatch(T, X, W, ResultType), CurrentResults),
    append(OldResults, CurrentResults, L).

% permutations of all such teams (results as described above)
% number of such teams: N
resultsPermutation(T, W, L, N, ResultType):-
    results(T, W, P, ResultType),
    length(P, N),
    permutation(P, L).

% wins/losses/draws as a call to results with 1/0/2 as ResultType (Win/Lose/Draw respectively)
wins(T, W, L, N) :- resultsPermutation(T, W, L, N, 1).
losses(T, W, L, N) :- resultsPermutation(T, W, L, N, 0).
draws(T, W, L, N) :- resultsPermutation(T, W, L, N, 2).

% List of scores (TeamScore) of a Team on a specified Week
getScore(Team, TeamScore, Week) :-
    match(Week, Team, TeamScore, _, _).
getScore(Team, TeamScore, Week) :-
    match(Week, _, _, Team, TeamScore).

% List of concedes (TeamConcede) of a Team on a specified Week
getConcede(Team, TeamConcede, Week) :-
    match(Week, Team, _, _, TeamConcede).
getConcede(Team, TeamConcede, Week) :-
    match(Week, _, TeamConcede, Team, _).

% sumList(List, Total)
% sum of the elements of the List (Total)
sumList([], Total) :-
    Total is 0.
sumList([Head|Tail], Total) :-
    sumList(Tail, Partial),
    Total is Head + Partial.

% Sum of scores (S) made by team T, on W.
scoreWeek(T, W, S) :- % scores in the week, W.
    findall(X, getScore(T, X, W), Scores), % get scores for each week
    sumList(Scores, S). % sum scores for each week

% Sum of concedes (C) of team T, on week W.
concedeWeek(T, W, C) :- % concedes in the week, W.
    findall(X, getConcede(T, X, W), Concedes), % get concedes for each week
    sumList(Concedes, C). % sum concedes of each week
```

```prolog
% Sum of scores (S) made by team T, in and before the week W.
scored(_, 0, 0).
scored(T, W, S) :- % scores in and before the week, W.
    team(T, _),
    W > 0,
    Old is W - 1,
    scored(T, Old, OldScore),
    scoreWeek(T, W, CurrentScore),
    S is OldScore + CurrentScore.

% Sum of concedes (C) made by team T, in and before the week W.
conceded(_, 0, 0).
conceded(T, W, C) :- % concedes in and before the week, W.
    team(T, _),
    W > 0,
    Old is W - 1,
    conceded(T, Old, OldConcede),
    concedeWeek(T, W, CurrentConcede),
    C is OldConcede + CurrentConcede.

% Average (A) of team T, in and before the week, W
% A = Scores - Concedes
average(T, W, A) :- % average in and before the week, W.
    scored(T, W, Scores),
    conceded(T, W, Concedes),
    A is Scores - Concedes.

% select team Team, that has a better average than any other team in BiggerTeams, on Week.
isLower(Team, Week, BiggerTeams) :-
    team(Other, _),
    \+ member(Other, [Team|BiggerTeams]),
    average(Team, Week, TeamAverage),
    average(Other, Week, OtherAverage),
    TeamAverage < OtherAverage.

isBiggest(Team, Week, BiggerTeams) :-
    team(Team, _),
    \+ member(Team, BiggerTeams),
    \+ isLower(Team, Week, BiggerTeams).

% getOrdered(TeamList, Week, N)
% get N teams with highest average on Week.
% TeamList goes from low to high (reversed).
getOrdered([], _, 0).
getOrdered([Head|Tail], Week, N):-
    N > 0,
    NextN is N - 1,
    getOrdered(Tail, Week, NextN), % recursively, get N - 1 teams, first.
    findall(Team, isBiggest(Team, Week, Tail), BigTeams), % get next Biggest Teams.
    % (BigTeams is a list since more than one team can have the same next biggest average.
%   write('Big Teams: '), write(BigTeams), write('  N: '), write(N), nl,
    member(Head, BigTeams). % get one team from the BigTeams.

% a sorted list and another list of same elements but without duplicates
% only use with sorted list
removeDuplicates([],[]).
removeDuplicates([Head|Tail], Unique):-
    Tail = [Head|_],
    removeDuplicates(Tail, Unique).
removeDuplicates([Head|Tail], [Head|Unique]):-
    Tail \= [Head|_],
    removeDuplicates(Tail, Unique).

% get an available week. (one with matches)
% get that week for once (no duplicate week)
getWeek(Week):-
    findall(W, match(W,_,_,_,_),AllWeeks),
    sort(AllWeeks, SortedWeeks),
    removeDuplicates(SortedWeeks, UniqueWeeks),
%   write(UniqueWeeks),
    member(Week, UniqueWeeks).

% get the ordered list of teams (L) by their averages on week W.
order(L, W):-
    getWeek(W),
    findall(X, team(X, _), AllTeams), % get the number of all teams
    length(AllTeams, N), % N is the number of all teams
    getOrdered(ReversedList, W, N), % get N teams ordered by their average
    reverse(ReversedList, L). % reverse it to get the highest-to-lowest list

% get the ordered list of top three teams (L) by their averages on week W.
topThree(L, W):-
    getWeek(W),
    findall(X, team(X, _), AllTeams), % get the number of all teams
    length(AllTeams, MaxN), % MaxN is the number of all teams
    N is min(MaxN, 3), % return all teams if there are less then 3 teams
    getOrdered(ReversedList, W, N), % get N teams ordered by their average
    reverse(ReversedList, L). % reverse it to get the highest-to-lowest list
```