LINKED LIST

PROBLEM DEFINITION:
The aim of the project is to create a data structure namely a linked list of information related to a football league, so that it's possible to search for required information from the data structure directly.

METHODS AND SOLUTION:
The source code is divided into the functions:

1. main()
2. print_vec()
3. print_main_list()
4. find_player()
5. compare_list()
6. period_with_max_goals()
7. max_goal_scored_players()
8. hat_trick_scored_players()
9. get_team_list()
10. get_player_list()
11. print_matches_goals()
12. print_matches_asc()
13. print_matches_desc()
14. compare_set_asc()
15. compare_set_desc()

and classes:

1. away_info
2. player

The **main()** function takes care of:

1. Processing of input arguments.
    a. There will be three input arguments to the executable, namely:
        i. Input data related to football league. Eg: input.txt
        ii. Operations to be done on linked list. Eg: operations.txt
        iii. Output file. Eg: output.txt

    ```
    $ ./ assignment3 input.txt operations.txt output.txt
    ```

    The above line is the command line execution of the executable 'LinkedList'.
2. Processing of input information.
    a. We open the input.txt file and read in line by line and for each line, we split it into tokens separated by comma and collect them into a vector.
    b. We store the player related information into separate objects of player class, but before inserting this new node into the main linked list, we check for existence of this player node in the main linked list. If not present, we create the new player node and there by the away team information too. If the player node is already present, we simply append the away team information to the existing player node.

    c. Now, we have the main linked list and then we sort it according to names of the players. Next, we read in the operations.txt file and collect the information into another vector.
3. Calling the necessary functions.
    a. Then we simply call the different functions for each operation on the now available linked list.

The **print_vec()** function:
1. This is a utility function for printing a vector of strings. This prints data to the output file.

The **print_main_list()** function:
1. This is a utility function for printing the main linked list for debugging purposes. This won't print anything to the output file.

The **find_player()** function:
1. This searches the main linked list built up to that point, if the given player has a node already in the main linked list. Return false, if the player doesn't have a node associated with his name.

The **compare_list()** function:
1. This is simply a comparison function for sorting the player nodes in the main linked list according to the player name.

The **period_with_max_goals()** function:
1. This gives the period in which there were maximum number of goals.
2. We simply count the number of goals for all players in two different periods and then compare them.

The **max_goal_scored_players()** function:
1. This gives the names of the players who scored the maximum number of goals.
2. We simply collect the player names along with corresponding number of goals they scored and them give the final vector of players, who scored maximum. The situation when more than one player has scored maximum goals is also taken care of.

The **hat_trick_scored_players()** function:
1. This gives the names of the players who scored a hat-trick.
2. Similar to max_goal_scored_players() function, we collect the player names. But now we collect the match ids that the players played in.
3. If the player had scored three goals in matches with same id, then that player is added to the return vector.

The **get_team_list()** function:
1. This gives the list of all teams in the football league. We collect all the team names and away team names and then make this list unique by loading into a set and return that final list.

The **get_player_list()** function:

1. This gives the list of all players in the league. We collect the player names in the main linked list and then return this final list after making the list unique.

The **print_matches_goals()** function:

1. This prints the away team names, minute of the goals and match ids for all the given vector of players.
2. We basically loop over all the player nodes in the main linked list and for required players, we collect the required information and then finally dump them to the output file.

The **print_matches_asc()** function:

1. This prints the given players' names and the associated match ids in a sorted fashion, sorted according to the match ids in ascending order.

The **print_matches_desc()** function:

1. This is similar to the print_matches_asc() function, except now the order of the match ids for sorting is descending.

The **compare_set_asc()** and compare_set_desc() functions:

1. These are the comparison functions for sorting operation used by print_matches_asc() and print_matches_desc() functions respectively.

The **away_info** class:

1. Stores the information related to away teams for each individual player. This is stored in a doubly linked list.

The **player** class:

1. Stores information related to each player and thereby all the away teams associated with that particular player. This is the main linked list, which contains the player name, team name and a doubly linked list, in which each node corresponds to one away team associated with this particular player.

FUNCTIONS IMPLEMENTED:

1. Dynamic allocation using new() for the player nodes and the away_info nodes in the two linked lists.
2. Used file streams, STL containers: vector, list, set, string and the Boost libraries: string/split and string/trim for splitting and trimming off the strings and the intrusive list for doubly linked list.
3. Necessary commenting in the source code.