Hacettepe University Computer Science Department

BBM 203 – Software Laboratuary Assignment 3 - Report

Name: Mert

Surname: Çökelek

ID: 21727082

E–mail: b21727082@cs.hacettepe.edu.tr

Subject: A basic football season simulation with C++ language.

Due Date: 16.12.2018

Advisor: Merve Özdeş

Execution Note:

I created a "Makefile" for the compilation to be easy.

The commands for execution:

```
>> make

>> ./a.out input.txt operations.txt output.txt
```

1- What is the problem in this assignment and what needs to be understood?

In this assignment the main task is to create a simple simulation that contains footballers, football teams, and the matches between them.

We are expected to handle this task using "Linked List" structure. There should be 2 linked lists, which are holding informations about the footballers, and the goals that the footballers scored, respectively.

2. My approach to this problem is:

a. First, determine the inputs and the outputs of the program.

i. The inputs are:
   1. Input.txt
   2. Operations.txt

ii. And the output is:

1. Output.txt

b. *The purposes of inputs and outputs:*
   i. **Input.txt:** Contains informations about the footballers, such as their names, teams, the away teams, match ID's, minutes of goals.
   ii. **Operations.txt:** There are 3 lines in this file. Each line holds 2 footballer names, the first line is to print the matches of the given footballers. The and the third lines are to print the given footballers' names and the match ID's that they've played.
   iii. **Output.txt:** There are 8 different informations to be printed in this file.
      1. *The most scored half:* In which half, there have been more goals scored during the matches?
      2. *Top Goal scorer:* Which footballer(s) have scored more goals than the others?
      3. *Players who scored a hat-trick:* The players who scored more than 3 goals during the matches.
      4. *List of teams.*
      5. *List of footballers.*
      - 6, 7, 8. informations are printed according to the operations file.
      6. *The matches of the given 2 footballers.*
      7. *Match ID's in ascending order* of given footballers.
      8. *Match ID's in descending order* of given footballers.

c. How should the input files be used in the program?
   i. I determined to create 2 structures, one for the "goals" and one for the "footballers".
   ii.

```
struct goal{
    string awayTeam;
    int minOfGoal, matchID;
    goal *gNext, *gPrev;    // The next and prev pointers for linked list.

    // Constructors.
    goal()
    : awayTeam(""),
    minOfGoal(-1),
    matchID(-1),
    gNext(nullptr),
    gPrev(nullptr){}

    goal(string awayTeam, int minOfGoal, int matchID, goal* gNext, goal* gPrev)
    :awayTeam(awayTeam),
    minOfGoal(minOfGoal),
    matchID(matchID),
    gNext(gNext),
    gPrev(gPrev){}
};
```

   iii. The fields in "goal" struct:
      1. *awayTeam:* is the name of the away team of the scored goal.
      2. *minOfGoal:* is the minute of the scored goal.

3. *matchID:* is the ID of the match, which the goal was scored.
4. *gNext and gPrev pointers* are for the “goal doubly linked list”.
5. *And 2 constructors* for goal struct.

```
struct footballer{
    string name, teamName;
    goal* GoalList;  // Pointer of the "Goals Linked List" of the footballer.
    footballer* nextFootballer; // The next pointer for linked list.

    //Constructors
    footballer():
    name(""),
    teamName(""),
    GoalList(nullptr),
    nextFootballer(nullptr)
    {}

    footballer(string n, string tn, goal* g, footballer* f) :
    name(n),
    teamName(tn),
    nextFootballer(f),
    GoalList(g)
    {}
};
```

iv. The fields of footballer struct:
    1. *name:* Name of the footballer
    2. *teamName:* Name of the team of the footballer
    3. *goalList:* is a pointer to a linked list, holding goals of the footballer.
    4. *NextFootballer:* is pointer to the next footballer, for the “footballer linked list”.
    5. *And 2 constructors* for footballer struct.

3.After making a general scheme of the work, my detailed algorithm for this problem is:

1. Read input.txt file.
    a. For this purpose, I defined a function.
    b. footballer* readInputFile(fstream &file, footballer* footballers):

        This function takes 2 parameters: 1 for the file pointer, other for the linked list pointer. After reading txt file and initializing the linked list, returning the updated linked list.

        In first while loop, I get the line and in the second while loop, I split it into pieces, seperated by commas. (”,”). After splitting, I get 5 different pieces named “value” for each line. I store them in an array named “currentFootballer[]” temprorarily.

And I start creating new struct variables(footballers). If there is no footballer created before, I create a new one, else, I update the linked list.

The 5 pieces are corresponding to name, teamname, away team, minute of goal, match ID respectively. With these informations, I also create the goal linked lists for each footballer. To create new footballer or new goal variable, I use the constructors I defined in the structure.

2. Read Operations.txt file.
    a. For this purpose, I defined a function.
    b. **void readOperations(fstream &file, footballer* footballers)**

This function also takes 2 parameters, one for the operations file and the other for the linked list holding footballers, I created in the previous function.

Similarly, I read the operations file with 2 while loops, one for each line and one for splitting every line by commas.

There are 3 lines in this file.

For each line, there is a unique operation.

Every single line contains two different footballer names,

To get the footballers by their names, I created a function.

Footballer* getFootballerByName(string name, footballer* list)

This function is searching for the footballer by its name recursively. Checks after every call, and finally finds the correct one.

In the first line, we are expected to print the footballer's matches, their goals, away teams, match ID's and the minutes of the goals.

For this purpose, I created a function.

void printMatchesOfFootballer(footballer f, ofstream file)

This function takes one parameter for the footballer, which was returned by the "getFootballerByName" function, and prints the informations of the given footballer.

void ascendingOrderAccordingToMatchID (footballer f, ofstream file)

void descendingOrderAccordingToMatchID (footballer f, ofstream file)

These 2 functions do similar operations.

They print the given footballer's matches sorted by ID's, ascending and descending, respectively. To do this, in while loop, iterating over the footballer's goal Linked list, and printing infos every time.

3. For creating Linked lists for footballers and goals mentioned @entry 1, I created 2 functions.

```
goal* insertGoal(goal* root, goal* newGoal){
    if(root == nullptr){                                    // If no goal defined before, create a new Linked List.
        root = new goal(newGoal->awayTeam, newGoal->minOfGoal, newGoal->matchID, nullptr, nullptr);
        return root;
    }else if(root->matchID > newGoal->matchID){     // If we need to add to head, since matchID is less than the head's.
        goal* newRoot = new goal(newGoal->awayTeam, newGoal->minOfGoal, newGoal->matchID, root, nullptr);
        root->gPrev = newRoot;
        return newRoot;
    }else{                                                                          // Add to the end or to the middle.
        goal* iter = root;                                  // Iterating over the Linked List to find the place to insert.
        goal* newElem = new goal;                                                   // The new goal, which will be inserted.

        newElem->matchID = newGoal->matchID;
        newElem->minOfGoal = newGoal->minOfGoal;
        newElem->awayTeam = newGoal->awayTeam;

        while(iter->gNext != nullptr && iter->gNext->matchID < newGoal->matchID)    // Find the place to insert orderly.
            iter = iter->gNext;

        if(iter->gNext == nullptr){                                                 // The goal will be inserted to the end.
            iter->gNext = newElem;
            newElem->gPrev = iter;
            newElem->gNext = nullptr;
            return root;

        }else{                                                  // The goal will be inserted in somewhere in the middle.
            newElem->gNext = iter->gNext;
            iter->gNext->gPrev= newElem;
            iter->gNext = newElem;
            newElem->gPrev= iter;
            return root;
        }
    }
}
```

There are 2 + 2 = 4 different situations in this process to create an ordered doubly goal linked list.

1. If we need to add to the head of the linked list;
    i. If the linked list is empty, create a new linked list and add the new goal to the head.
    ii. If the adding element's Match Id is less than the head's, we need to add to the head, link the next pointer to the old head, link the old head's prev pointer to new head, and return the new head.
2. In this circumstance, there are 2 different situations.
    i. Adding to the end, since the new ID is the largest one in the linked list. To do this, we should iterate on the linked list to find the correct place to add. Untill the next pointer is not null and the iterator's ID is less than the new ID, go next.
    ii. If it is the end of the linked list, put it to the end, link it's next to null, prev to the current linked list's tail.

iii. If it's going to be added into the middle, the links are updated, returned root.

```
footballer* insertFootballer(footballer* root, string name, string teamName, goal* goal){
    if(root == nullptr){                            // If Linked List is empty, create a new footballer and return it.
        root = new footballer(name, teamName, goal, nullptr);
        return root;
    }else if(root->name == name){            // If the footballer has been registered before and has the smallest name.
        root->GoalList = insertGoal(root->GoalList, goal);
        return root;
    }else if(root->name > name){        // If the new footballer's name is smaller than the head's name, add it to head.
        footballer* newRoot = new footballer(name, teamName, goal, root);
        return newRoot;
    }else{                                                          // If we need to insert in the middle or the end.
        footballer* iter = root;                        // Iterate over the linked list to find a suitable place to add.

        while(iter->nextFootballer != nullptr && iter->nextFootballer->name < name)
            iter = iter->nextFootballer;

        footballer* temp;
        if(iter->nextFootballer == nullptr){                                // Inserting at the end of the Linked List.
            temp = new footballer(name, teamName, goal, nullptr);
            temp->nextFootballer = nullptr;
            iter->nextFootballer = temp;
        }else if(iter->nextFootballer->name == name) { // If the footballer is registered before, just update goal list.
            temp = iter->nextFootballer;
            temp->GoalList = insertGoal(temp->GoalList, goal);
        }else{                                                                  // Inserting in the middle.
            temp = new footballer(name, teamName, goal, nullptr);
            temp->nextFootballer= iter->nextFootballer;
            iter->nextFootballer= temp;
        }
        return root;
    }
}
```

To create a footballer linked list ordered, there are 3 + 3 = 6 different possibilities.

1. If we are adding to the head and the linked list is empty, create a new node and link its next to null, initialize it, return it.
2. If it's not null but the head's name is equal to the inserting element's name, then update the goal linked list of the current footballer.
3. If it's not null and the head's name is greater than the inserting element's name, then create a new footballer and link it's next to the old head. Return the new head.
4. If we are adding in the middle of the linked list or the end of it, in w while loop, iterating over the linked list until reaching a bigger element.
    a. If the new element's name is the larger, add it to the end.
    b. If in the linked list, the same named footballer was registered before, update its goal statics.
    c. For the last possibility, we are adding a new element to somewhere in the middle of the linked list.
    d. Return root.

The last function in my program is:

void printOutput(footballer* footballers, ofstream file):

This function takes the footballers linked list as parameter.

There are 5 different writings for this function.

1. **Most Scored Goal Half:** There are 2 halfs in a football match and in the input file, the minutes of the goals are given, we need to sum them determine in which half, more goals have been scored.

   To do this, my program iterates over the linked lists, and holds two counters named "goalsInfirstHalf" and "goalsInSecondHalf", incrementing them when it's time. After this, comparing these two integers and determining in which half more goals have been scored during the matches.

2. **Top Scorer:** In this part, I need to print the footballer(s), who have scored the most goals. To determine this, I declared a field named "goals" in footballer struct, and every time updating the footballer's goal list, incremented this variable by 1. Finally, made a comparison between the other footballers, and printed the most scored one.

3. **The footballers who made hat-trick:** In this part, we are expected to print the footballers who scored more than 3 goals in one match. To do this, created a string for holding the footballer names and iterated over the linked lists, and counted every goals per match for every single footballer. Finally, printed the footballers who made hat-trick.

4. **List of Teams:** Prints out the team names in the season. To do this, created a string for holding the team names and with while loop, iterated over the linked list of the footballers, every time meeting a new team name, appended it into the string. Finally, printed the string.

5. **List of Footballers:** Prints all the footballers' names. To do this, created a string for holding the names, and with while loop, iterated over the footballers linked list, every time meeting a new name, appended that name into the string. Finally, printed the strind.