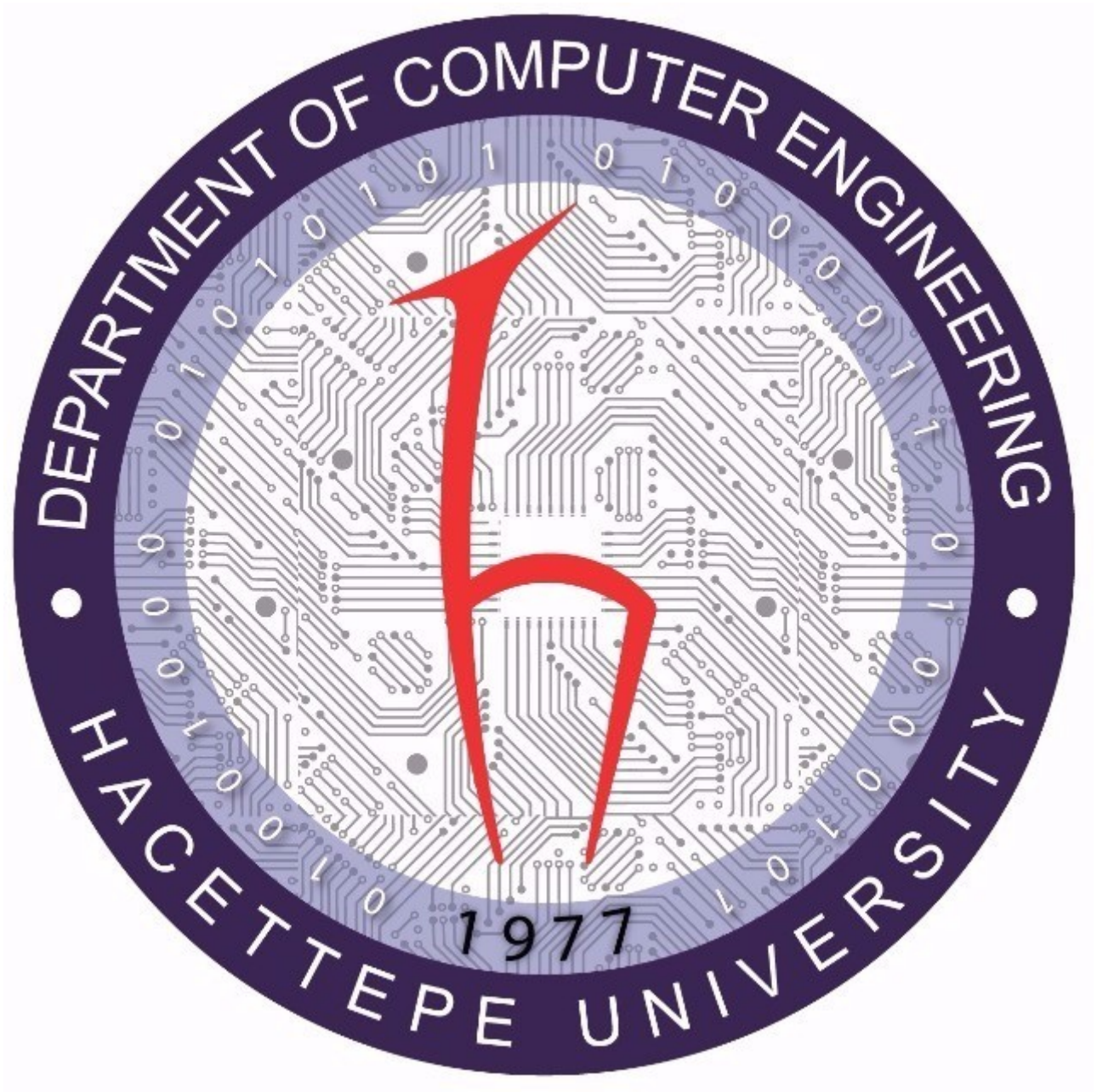

GÖKHAN ÖZELOĞLU

21627557

BBM 203 – DATA STRUCTURES

ASSIGNMENT 3

2. SOFTWARE USING DOCUMENTATION

2.1. Software Usage

My program must be compiled with **C++11** version. So, I created a **Makefile**. Compiling and running of the program as follows:

```
$ make
```

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

Makefile compiles the program in **C++11** version. So, I avoided that any compile error. After compiling code, executable file is being created name as **a.out**. Other 3 input files' names can be changed. Different file names can being run in this program.

input.txt : This file includes information about the footballers. Their names, team, away teams which they scored, minute of the goal and match id. Each feature separated by commas.

operations.txt : This file consist of 3 lines and 2 names for each line. The first line represents 6th calculation in output file. The second line represents 7th and the third line represents 8th line.

output.txt : This file is being created after program execution. There are 8 different titles that calculated by program.

2.3. Error Messages

There is not any special error situations or messages in this assignment. Nevertheless, I give error messages if the input files cannot open. This error messages are printed out the screen.

3. SOFTWARE DESIGN NOTES

3.1. Description of the Program

3.1.1. Problem

In this assignment, we have 2 different input files which names are <input.txt> and <operations.txt>. Input file have information about the league and footballers. The program produces some of outputs related to operation file and pdf. We should have code with singly linked list and doubly linked list. While storing the footballers, we must have used singly linked list. Also, we must have used doubly linked list, while storing matches. Moreover, footballer names and match ID's must be sorted in ascending order.

3.1.2. Solution

I used *object-oriented programming* concepts in this assignment. I defined 3 different class which names are **Match**, **Footballer** and **Team**. Main classes are **Match** and **Footballer**. Team class is being defined for listing team list. I had to store teams another list because teams should not have been stored more than one time. No duplicate teams. Footballer class have 2 strings and 2 pointers as a variable. Strings stores footballer's name and his team name. Pointers show next footballer node and his match nodes. I defined footballer name and his team in **private**. So, I defined 2 **accessor** methods to reach them. Moreover, it includes 2 different constructor to create new nodes and pointer that shows node. One constructor is **default constructor** and other one is **parameterized constructor**. It takes 2 different parameters which are footballer and team name. Also, next footballer node pointer and match node pointer points to *NULL* in initial time. Furthermore, I defined **destructor** method to free the class. Other two functions are adds the new footballer into footballer linked list and finds the period of the all matches which is scored in the league.

The second class is **Match** that includes *away team*, *match id* and *goal minute* as private variable. Match linked list must have been *doubly linked list*, so it must points to the next and previous match node. This class has one constructor to create new nodes while adding on linked list. So, the constructor takes 3 parameters which are *away team, minute* and *match id*. Variables are assigned in the constructor. In addition, next and previous match pointers points to **NULL**. *Destructor* frees the class objects when program finish. Moreover, I am accessing private variables with *accessor* methods. Finally, the class has a function that adds the new matches.

I tried to divide my program and problem in functions. So, it made my code more *modular, readable* and *maintainable*. I call the functions from **main()** function in proper time.

3.4. Algorithm

My program starts with reading files line by line. While reading input file, program calls the function which adds the new footballers into linked list. The footballers are adding in alphabetic ordered. After reading input file, program starts to produce outputs. Firstly, finds the period with calling related function. Secondly, prints out the top goal scorer with calling function again. Then, finds the hat-trick scored players. If there is no such a player, prints out false. After that, by using **Team** class, teams are listed and prints out. Then, footballers are printing out in alphabetic order. With the 6th title, the program starts to reading operation file. Splits the each line and calls proper functions. Finally, the program closes the files and deletes the pointers. Frees all of the linked list nodes.

Functions

```
Match* Match::addNewMatch(Match *match_node, string away_team, int minute, string match_id)
```

Adds the new match nodes into linked list. Firstly, controls the **head** pointer is *NULL* or not. If it is *NULL*, then creates the new player node. If it is not NULL, Creates two Match nodes that one of them is pointing to current node and other one is pointing to previous node. I am holding previous node because some of nodes are inserting to middle of the linked list. I have to link new node with next and previous node. I added 2 if statement for 2 special situations. If the match linked list have one match node, I controlled this and I added the node in this if statements. If statement added the new node as a second node and else if statement added the new node as a first node. Finally, if linked list have more than 2 nodes, I added them in else situation. **while** loop goes until NULL and each iteration it controls the match ID's. If the new match id is small or equal than temp node, the program adds the new node in **while** loop. In each iteration, previous and temp match nodes are changing.

```
Footballer* Footballer::addNewFootballer(Footballer *head, string name, string team_name, string away_team, int minute, string match_ID)
```

This function is so similar with adding new match function. It controls the **head** node is *NULL* or not. There is some special cases while adding footballers. So, firstly, I handled them by controlling *if-else* statements. I faced some problems while adding the first 3 footballers in input file. After adding 3rd footballer, adding operation continue with while loop. In the while loop, there are some if statements that controls the names. I left comment lines on the code for each situations.

```
int Footballer::findPeriod(Footballer *head)
```

This function just finds which period is the most scored in the league. Counts the goals with 2 different integer variables for each periods. Program traverse the footballer and each footballer's linked list with while loop. So, I constructed nested struct. The outer-while loop traverse the footballers and inner-loop traverse the each footballer's match linked list. The program updates the footballer and match nodes each iteration. Finally, returns the integer value for **1** for second half, **0** for first half and **-1** for equal situation.

```
Team* Team::addTeamsToList(Team *head, string team_name)
```

This function adds the teams only once time to print out the team list. Controls the team is already added before or not. If not added, it adds and return the new list. If it is added, return the list without adding.

```
void printGivenFootballer(fstream& outputFile, Footballer *head, string name)
```

That writes the information about the given footballer with traversing the linked list. This function writes directly on the output file.

```
void printAscendingMatchID(fstream& outputFile, Footballer *head, string name)
```

This function was code for **#7**output. Matches were added in ascending order, so the function writes the output by traversing the matches linked list.

```
void printDescendingMatchID(fstream& outputFile, Footballer *head, string name)
```

This function is so similar with previous function. In this case, it writes the matches in descending order. So, firstly, I traverse the linked list until *NULL* pointer. Then, I came back to head by using **prev node**.

```
void printTopGoalScorers(fstream& outputFile, Footballer *head)
```

I used **map** in this function while storing the each footballer's goal scores. Firstly, I counted each footballer's goal scores and added them into map. Then, I traverse the map by using **iterator**. After found top goal score, I traverse the map again to write output.

```
void printHattrickScorers(fstream& outputFile, Footballer *head)
```

This function searches any footballer scored hat-trick or not. If there is any player who scored hat-trick, the function writes his name on the output file. The goals are counting with **counter** variable and comparing with match ID's.

NOTE : I do not have any **NOT IMPLEMENTED** function.