Middle East Technical University Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring '2015-2016

Take Home Exam 1

Due date: 24 March 2016, Thursday, 23:55

1 Objectives

This homework aims to familiarize you with functional programming concepts and Haskell basics.

2 Problem Definition

In this homework, you are going to implement functions that operate on a user and group network called 'DB', which is an over-simplified social network. 'DB' structure consists of two lists, one for users and one for groups. Data definitions are given as below:

```
type RealName = String
type UserName = String
type GroupName = String
type Message = String

data Post = Post UserName Message deriving (Show, Eq)
data To  = UserID UserName | GroupID GroupName deriving (Show, Eq)
data User = User UserName RealName [UserName] [Post] deriving (Show, Eq)
data Group = Group GroupName [UserName] [Post] deriving (Show, Eq)
data DB  = DB [User] [Group] deriving (Show, Eq)
```

3 Functions

You will be implementing 2 families of functions in this homework, the first family is *Commands*. These functions are used to make changes in the `DB`, so they take a `DB` and return a new `DB` with required changes. Other family of functions is *Queries*. These functions take a `DB`, but they don't make changes to it, instead they extract information from it.

3.1 Commands

3.1.1 newUser

```
newUser :: DB -> User -> DB
```

This function adds the given user to `DB`.
If a user with the same user name already exists, this function should have no effect.

```
> newUser (DB [] []) (User "john" "John" [] [])
DB [User "john" "John" [] []] []
```

3.1.2 addFriend

```
addFriend :: DB -> UserName -> UserName -> DB
```

This function adds both user names to each others' friend lists.

```
> addFriend (DB [User "ssg" "Sedat" [] [],User "kanzuk" "Basak" [] []] []) "ssg↩
    " "kanzuk"
DB [User "ssg" "Sedat" ["kanzuk"] [], User "kanzuk" "Basak" ["ssg"] []] []
```

3.1.3 sendPost

```
sendPost :: DB -> UserName -> Message -> [To] -> DB
```

This function takes user name of the sender, a message and a list of recipients the new post to be delivered to and returns a new `DB` with the appropriate changes.

```
> sendPost (DB [User "mark" "Mark Zuckerberg" [] []] []) "mark" "hello world" [↩
    UserID "mark"]
DB [User "mark" "Mark Zuckerberg" [] [Post "mark" "hello world"]] []
```

3.1.4 newGroup

```
newGroup :: DB -> GroupName -> DB
```

This function creates a new group with the given group name and adds it to the `DB`. Newly created groups have no members and posts.

```
> newGroup (DB [] []) "METU"
DB [] [Group "METU" [] []]
```

3.1.5 addMember

```
addMember :: DB -> GroupName -> UserName -> DB
```

This function adds the user with the given user name to the group with the given group name. If the user already is a member of the group, this function should have no effect.

```
> addMember (DB [User "bgates" "Bill Gates" [] []] [Group "Microsoft" [] []]) "↩
    Microsoft" "bgates"
DB [User "bgates" "Bill Gates" [] []] [Group "Microsoft" ["bgates"] []]
```

3.1.6 removeMember

```
removeMember :: DB -> GroupName -> UserName -> DB
```

This function removes the user with the given user name from the group with the given group name. If the user is not a member of the group, this function should have no effect.

```
> removeMember (DB [User "bgates" "Bill Gates" [] []] [Group "Microsoft" ["↩
    bgates"] []]) "Microsoft" "bgates"
DB [User "bgates" "Bill Gates" [] []] [Group "Microsoft" [] []]
```

3.2 Queries

3.2.1 getFriendNames

```
getFriendNames :: DB -> UserName -> [RealName]
```

This function returns *real names* of the friends of the user with the given user name.

```
> getFriendNames (DB [User "ssg" "Sedat" ["kanzuk"] [], User "kanzuk" "Basak" [↩
    "ssg"] [] []) "ssg"
["Basak"]
```

3.2.2 getPosts

```
getPosts :: DB -> To -> [Post]
```

This function returns a list of posts the given user or the group *received*.
Note that when a user is removed from a group, posts which were previously seen by the user are still visible to the user.

```
> getPosts (DB [User "mark" "Mark Zuckerberg" [] [Post "mark" "hello world"]] ↩
    []) (UserID "mark")
[Post "mark" "hello world"]
```

3.2.3 listGroups

```
listGroups :: DB -> UserName -> [Group]
```

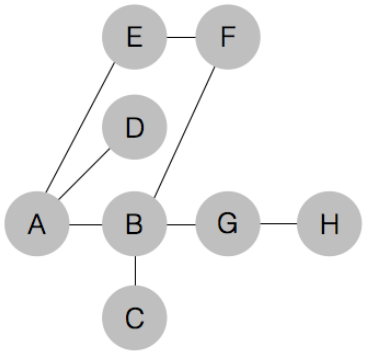
This function takes a user name, and returns a list of all the groups this user is a member of.

```
> listGroups (DB [User "bill" "Bill gates" [] [],User "joe" "Johnny" [] []] [↩
    Group "microsoft" ["joe","bill"] [],Group "metu" ["joe"] []]) "joe"
[Group "microsoft" ["joe","bill"] [],Group "metu" ["joe"] []]
```

3.2.4 suggestFriends

```
suggestFriends :: DB -> User -> Int -> [User]
```

This function suggests friends for a user with the given user name. The second parameter is the minimum number of common friends for a suggestion to be valid.
For example, assume we are suggesing friends to user A with 2 common friends.



Here, since A and C has only B as a common friend, it will not be suggested. But since A and F has 2 common friends (B, E), it should be suggested.
For the same friendship graph, if the required common friend count was 3, no friends would be suggested.
Also, for the same friendship graph, if the common friend requirement was 1, F, C and G would be suggested but H wouldn't be.
Also, an existing friend must not be suggested!

```
> suggestFriends (DB [(User "a" "A" ["e", "d", "b"] []), (User "e" "E" ["a", "f↩
    "] []),(User "d" "D" ["a"] []),(User "b" "B" ["a", "f", "c"] []),(User "c" "↩
    C" ["b"] []),(User "f" "F" ["e", "b"] []),(User "g" "G" ["b", "h"] []),(User↩
     "h" "H" ["g"] [])] []) (User "a" "A" ["e", "d", "b"] []) 2
[User "f" "F" ["e","b"] []]
```

4 Submission

- You will upload a single Haskell source file named `HW1.hs`.
- Submission will be done via COW.
- **Late submission**: Regulations in the course website[1] applies.

5 Grading

- Functions will have different weights in grading based on their relative difficulties.
- We will be using `ghc` on *inek* machines, make sure your code works in them. Otherwise, you will not be able to modify your code.

[1] `http://www.ceng.metu.edu.tr/course/ceng242/syllabus#Grading_and_other_policies`

- While compiling your code, we will be using the following command: `ghc --make main.hs` An example `main.hs` file will be provided in COW.

6 Regulations

- **Programming Language:** Haskell
- You are not allowed to import any module.
- **Cheating:** We have zero tolerance policy for cheating. In case of cheating, all parts involved (source(s) and receiver(s)) get zero. People involved in cheating will be punished according to the university regulations.
- Remember that students of this course are bounded to code of honor and its violation is subject to severe punishment.
- **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.
- Do not modify the given type definitions.
- Failing to follow the homework submission rules and these regulations will result in grade reduction.