CSE 232 Spring 2018

Programming Project 07

Assignment Overview
This assignment is worth 50 points (5% of the course grade) and must be completed and turned in before 11:59pm on Monday, March 19th, 2018.

Problem Description
In this project, you have to manage a list of computer servers, where each server hosts multiple users. The server data is stored as an associative data structure type called `ServerData` which is an STL `map<string, set<string>>`. Thus, the `keys` of this map are strings that represent server names, and the `values` associated with each server name is an STL set of strings that are the names of users on that server.

You will have to write functions that add a user to a server, disconnect a user from a server, balance the server load, etc.

Program Specifications
You are provided with a `proj07_functions.h` that contains the signatures of all the functions you must implement. You must implement those functions and turn in your results as `proj07/proj07_functions.cpp` to Mimir. You are also provided with `proj07_main.cpp` that does some minimal testing of these functions. Of course, the real test cases are on Mimir as always.

Functions
`bool AddConnection(ServerData &sd, ServerName sn, UserName un)`
This function tries to connect the user `un` to the server `sn`. If the server exists, it adds `un` to that server. If the server does not exist, it is created (added to `sd`) and the user `un` is added to that server. Both of those conditions return `true`

If the user `un` is already connected to the server `sn`, then no action is taken, and the function returns `false`.

`bool DeleteConnection(ServerData &sd, ServerName sn, UserName un)`
This function tries to disconnect the user `un` from the server `sn`, and returns `true` if `d` is successfully updated with that information. If the user `un` is not connected to the server `sn`, then no action is taken, and the function returns `false`.

`ServerData ParseServerData(const std::string &fname)`
`fname` is the name of a file that contains an arbitrary number of lines of text where each line consists of three strings, each space separated. Each line is either of two forms:

- `user_name join server_name`
- `user_name leave server_name`

For each line in this file, if the 2nd string is `join`, then add `user_name` to `server_name`. If the 2nd string is `leave`, remove user name from server name. If specified action cannot be performed, e.g. if trying to add `user_name` to `server_name` when `user_name` is already connected to `server_name`, then that action is simply ignored.

Errors on `ParseServerData`

- if the file name cannot be opened, the function throws an `invalid_argument` exception
- if any of the input lines has a command other than `leave` or `join`, the function throws a `domain_error` exception

`void PrintAll(std::ostream out&, const ServerData sd&)`
Prints the contents of `sd` to `out` (not `cout`, the passed ostream `out`). Format is:

```
server_name : user_name user_name … user_name
server_name : user_name user_name … user_name
```

There is a `'\n'` after each line including the last

`set<string> AllServers(const ServerData & sd)`
Returns a `set<string>` which is the set of all servers in the `sd`

`set<string> AllUsers(const ServerData & sd)`
Returns a `set<string>` which is the set of all users on any server in the `sd`

`set<string> HasConnections(const ServerData &sd, UserName un)`
This function returns a `set<string>` of all the server names in `sd` that the user `un` is currently connected to. Note that the return set could be empty.

`set<string> HasUsers(const ServerData &sd, ServerName sn)`
This function returns a `set<string>` of all the user names in the server `sn` that are currently connected in `sn`. Note that the return set could be empty.

`void BalanceServers(ServerData &sd, ServerName sn1, ServerName sn2)`
This function tries to balance the number of users that are connected to server `sn1` and `sn2`. If a user is connected to both servers, then they are ***not*** moved. All the users that are only connected to one of the 2 servers are moved in the following fashion. These users are sorted alphabetically by their name, and the first half are moved to server `sn1`, and the other half are moved to server `sn2`. (Note: if there are an odd number N of users, then N/2 +1 users are moved to `sn1`.)

`void CleanAndBalance(ServerData &sd)`
This function first removes all duplicate users from `sd`. Then the unique users are moved around in the following way.

1. All the user names and server names are sorted alphabetically
2. The users are distributed to servers in a round-robin fashion in alphabetical order of server and user names

For example, assume there are 3 servers named `x,y,z` and 8 users named `a,b,c,d,e,f,g,h`. Note we already sorted both lists (which you would have to do).

- users `a,d,g` will be moved to server `x`, users `b,e,h` will be moved to server `y` and users `c,f` will be moved to server `z`

Deliverables

`proj07proj07_functions.cpp` -- your completion of the functions described above. Only `proj07/proj07_functions.cpp` is turned in to Mimir.

1. Remember to include your section, the date, project number and comments.
2. Please be sure to use the specified directory and file name.

Manual Grading

In lab05 you worked with Code Style. The TAs will apply the things that you learned in your Code Review lesson on the code you submit. 4 points are reserved (out of 104) for this.

Assignment Notes

1. Mimir allows us to test the functions in `proj07_functions.cpp` individually and we will do that.
2. We provide a `pro07_main.cpp` to test your code which you can modify for your own testing, but only `proj07_functions.cpp` needs to be turned into Mimir
3. `proj07_functions.h` is provided in the project07 directory. It will be used in testing. When we do testing, we will use the file we provide. If you change and submit your own version, Mimir will ignore your version and use the file originally provided.
4. Algorithms/iterators are your friends here! Check out the following:
    a. `set_intersection`, things that are common in two sets
    b. `set_symmetric_difference`, opposite of intersection (unique elements from each set).
    c. `copy_n`, copy n elements from one set to another
    d. the `map.find` function (different from the regular find, utilizes ordered search)
    e. `inserter` iterator, adds elements to a map/set
    f. be on the lookout for others as well. Let the algorithms do the work!