

1 Hints for Task A

Define `Profile` class. A profile is similar to a person, but is used in the world of network.

1. Data members are `username` and `displayname`. Since they are private, need to provide methods (aka operations) in public interfaces to access or modify these data members.
2. Constructors are used to initialize data members and create an object with needed operations – for example, getters or accessors like `getUsername`, `getFullName`, setters or mutators like `setDisplayname`.

```
1 #include <iostream>
2 using namespace std;
3
4 class Profile {
5 private:
6     string username;
7     string displayname;
8 public:
9     // Profile constructor for a user (initializing
10    // private variables username=usrn, displayname=dspn)
11    Profile(string usrn, string dspn);
12
13    // Default Profile constructor (username="", displayname="")
14    Profile();
15
16    // Return username
17    string getUsername();
18    // Return name in the format: "displayname (@username)"
19    string getFullName();
20    // Change display name
21    void setDisplayName(string dspn);
22 };
23
24 //TODO:
25 //(1) set data member username by formal parameter usrn,
26 //(2) set data member displayname by formal parameter dspn.
27 Profile::Profile(string usrn, string dspn) {
28
29
30 }
31
32 //TODO:
33 // Non-default Profile constructor
34 // (1) set data member username to be "",
35 // (2) set data member displayname to be "".
36 Profile::Profile() {
37
```

```

38 }
39 }
40
41 //TODO: Return username
42 string Profile::getUsername() {
43
44 }
45
46 //TODO: Return name in the format: "displayname (@username)"
47 string Profile::getFullName() {
48
49
50
51 }
52
53 //TODO: Change data member displayname by formal parameter dspn
54 void Profile::setDisplayname(string dspn) {
55
56 }
57
58 int main() {
59     Profile p1("marco", "Marco");
60     cout << p1.getUsername() << endl; // marco
61     cout << p1.getFullName() << endl; // Marco (@marco)
62
63     p1.setDisplayName("Marco Rossi");
64     cout << p1.getUsername() << endl; // marco
65     cout << p1.getFullName() << endl; // Marco Rossi (@marco)
66
67     Profile p2("tarma1", "Tarma Roving");
68     cout << p2.getUsername() << endl; // tarma1
69     cout << p2.getFullName() << endl; // Tarma Roving (@tarma1)
70 }

```

Sample output is as follows.

```

marco
Marco (@marco)
marco
Marco Rossi (@marco)
tarma1
Tarma Roving (@tarma1)

```

2 Task B

Define class **Network**, describing how a profile in this network follows another. It has the following data members.

- **profiles** is an array of **Profile** whose capacity is **MAX_USERS** with value 20. That is, a network has at most twenty profiles.
- **MAX_USERS** is a **static data member**, also called **class data member**. Think each network has exactly the same value for this data member, or, this data member is shared by all objects of **Network**. So, we need only one copy of this data member in the whole class.

A similar example is Person class. Suppose we only care about name, age, and the number of total population in the world. Each person has name and age, which differ from one individual person to another. However, the number of total population in the world should be set as static data member since this value is not changed from one person to another.

- `numUsers` stores the current size (number of profiles) of data member `profiles` in the network.
- Note that we use a static allocated array of profiles, so the size is a constant – `MAX_USERS` – and we use `numUsers` to specify the size of the array, or the actual number of elements in array `profiles`.
- Here is a sidenote describing the difference between static allocated array and dynamically allocated array. In static allocated array, the capacity (maximum number of elements allowed) must be a const. There is only one chance in compilation time to apply memory for the array, so we normally apply for the maximum number of elements we need for the application, aka, capacity. However, we may not use all the capacity of the array, so we need to use another variable called size to track the current number of elements in the array.

It is like when you apply to build a hotel, you would apply for the maximum of rooms allowed, that is call **capacity**, this needs to be a const. But the actual occupied rooms can be vary and is saved in **size**.

In contrast, the size of a dynamically allocated array does not need to be const. However, we need to use `new` to apply for memory and `delete` to release memory when we no longer need it.

In Project Minesweeper, we use dynamically allocate one-dimensional array `int* cells` whose **size** can be determined in run time. It is like dynamically allocated array is tailored and fit to the need exactly, no unused elements in the array.

Operator `*` in `int*` implies one or many. Variable of type `int*` can save the address of one int, it can also save the initial address of an array of ints. In Project Minesweeper, `cells` stores the initial address (ie, the address of the first element) in an array of ints.

In Project Game 1024, we use dynamically allocated two-dimensional array `int** panel`. Type `int**` saves the address of an array of `int*`, the size of this array is `numRows`, and each `int*` saves the address of an array of `int`, the size of that array is `numCols`.

Said differently, `panel` has `numRows` rows, each row points to (save the initial address) of an array of `numCols` ints. So `panel` is a two-dimensional array of ints.

```
1 #include <iostream>
2 using namespace std;
3
4 class Profile {
5 private:
6     string username;
7     string displayname;
8 public:
9     // Profile constructor for a user (initializing
10    // private variables username=usrn, displayname=dspn)
11    Profile(string usrn, string dspn);
12
13    // Default Profile constructor (username="", displayname="")
14    Profile();
15
16    // Return username
17    string getUsername();
18
19    // Return name in the format: "displayname (@username)"
```

```

20     string getFullName();
21
22     // Change display name
23     void setDisplayName(string dspn);
24 };
25
26 //TODO:
27 // Non-default Profile constructor
28 // (1) set data member username to be "",
29 // (2) set data member displayname to be "".
30 Profile::Profile(string usrn, string dspn) {
31
32
33 }
34
35 //TODO: Default Profile constructor (username="", displayname="")
36 Profile::Profile() {
37
38 }
39
40 //TODO: Return username
41 string Profile::getUsername() {
42
43 }
44
45 //TODO: Return name in the format: "displayname (@username)"
46 string Profile::getFullName() {
47
48
49
50 }
51
52 //TODO: Change display name
53 void Profile::setDisplayName(string dspn) {
54
55 }
56
57 class Network {
58 private:
59     static const int MAX_USERS = 20; // max number of user profiles
60     int numUsers; // number of registered users
61     Profile profiles[MAX_USERS]; // user profiles array:
62     // mapping integer ID -> Profile
63
64     // Returns user ID (index in the 'profiles' array) by their username
65     // (or -1 if username is not found)
66     int findID (string usrn);
67
68 public:
69     // Constructor, makes an empty network (numUsers = 0)
70     Network();
71
72     // Attempts to sign up a new user with specified username and displayname
73     // return true if the operation was successful, otherwise return false
74     bool addUser(string usrn, string dspn);

```

```

75 };
76
77 //TODO: set data member numUsers to be zero.
78 Network::Network() {
79
80 }
81
82 //TODO: find out whether usrn is in the profiles or not.
83 //If yes, return the index, otherwise, return -1.
84 int Network::findID(string usrn) {
85
86
87
88
89
90 }
91
92 //TODO: Attempts to sign up a new user with specified username and displayname
93 // return true if the operation was successful, otherwise return false
94 //The specified usrn needs to contain alphabet and digit letter only. That is, only 'A'-'Z', 'a'-'z',
    or '0'-'9' are allowed.
95 //Hint: use int isalpha ( int c ) to test
96 //whether character c is alphabetic, ie, 'a'-'z', or 'A'-'Z'.
97 //Similarly, use int isdigit ( int c ) to test
98 //whether character c is decimal digit '0' - '9'.
99 //To use isalpha or isdigit, need to
100 //include ctype library by
101 // #include <ctype>
102 bool Network::addUser(string usrn, string dspn) {
103     //TODO: if any character in usrn is
104     //neither alphabet nor digit, return false
105
106
107     //TODO: if numUsers is larger than equal to MAX_USERS,
108     //return false.
109
110
111     //TODO: if usrn is not an existing username,
112     //ie, usrn not found in profiles,
113     //add usrn to profiles, increase numUsers by one.
114     //
115     //hints: call findID method of Network class with usrn.
116     //If the return is -1, then usrn is not found in profiles.
117     //create a profile object with formal parameters
118     //usrn and dspn,
119     //and put that profile object to data member profiles,
120     //increase number of users, denoted by numUsers, by 1.
121     //Do not forget to return true.
122     if (findID(usrn) == -1) { //usrn not found in profiles
123
124
125
126     }
127     else //usrn is an existing user in profiles.
128         //TODO: what is the return?

```

```

129 }
130
131
132 int main() {
133     Network nw;
134     cout << nw.addUser("mario", "Mario") << endl; // true (1)
135     cout << nw.addUser("luigi", "Luigi") << endl; // true (1)
136
137     cout << nw.addUser("mario", "Mario2") << endl; // false (0)
138     cout << nw.addUser("mario 2", "Mario2") << endl; // false (0)
139     cout << nw.addUser("mario-2", "Mario2") << endl; // false (0)
140
141     for(int i = 2; i < 20; i++)
142         cout << nw.addUser("mario" + to_string(i),
143             "Mario" + to_string(i)) << endl; // true (1)
144
145     cout << nw.addUser("yoshi", "Yoshi") << endl; // false (0)
146 }

```

Sample output is as follows.

```

1
1
0
0
0
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
0

```

3 Task C

In addition to the code of Task B, define two more methods `follow` and `printDot` of `Network` class.

```

1 #include <iostream>
2 using namespace std;

```

```

3
4 class Profile {
5 private:
6     string username;
7     string displayname;
8 public:
9     // Profile constructor for a user (initializing
10    // private variables username=usrn, displayname=dspn)
11    Profile(string usrn, string dspn);
12
13    // Default Profile constructor (username="", displayname="")
14    Profile();
15
16    // Return username
17    string getUsername();
18
19    // Return name in the format: "displayname (@username)"
20    string getFullName();
21
22    // Change display name
23    void setDisplayName(string dspn);
24 };
25
26 //TODO:
27 // Non-default Profile constructor
28 // (1) set data member username to be "",
29 // (2) set data member displayname to be "".
30 Profile::Profile(string usrn, string dspn) {
31
32
33 }
34
35 //TODO: Default Profile constructor (username="", displayname="")
36 Profile::Profile() {
37
38 }
39
40 //TODO: Return username
41 string Profile::getUsername() {
42
43 }
44
45 //TODO: Return name in the format: "displayname (@username)"
46 string Profile::getFullName() {
47
48
49 }
50
51
52 //TODO: Change display name
53 void Profile::setDisplayName(string dspn) {
54
55 }
56
57 class Network {

```

```

58 private:
59     static const int MAX_USERS = 20; // max number of user profiles
60     int numUsers; // number of registered users
61     Profile profiles[MAX_USERS]; // user profiles array:
62                                     // mapping integer ID -> Profile
63
64     // Returns user ID (index in the 'profiles' array) by their username
65     // (or -1 if username is not found)
66     int findID (string usrn);
67
68     bool following[MAX_USERS][MAX_USERS]; // friendship matrix:
69     // following[id1][id2] == true when id1 is following id2
70
71 public:
72     // Constructor, makes an empty network (numUsers = 0)
73     Network();
74     // Attempts to sign up a new user with specified username and displayname
75     // return true if the operation was successful, otherwise return false
76     bool addUser(string usrn, string dspn);
77
78     // Make 'usrn1' follow 'usrn2' (if both usernames are in the network).
79     // return true if success (if both usernames exist), otherwise return false
80     bool follow(string usrn1, string usrn2);
81
82     // Print Dot file (graphical representation of the network)
83     void printDot();
84 };
85
86 Network::Network() {
87     //TODO: set data member numUsers to be zero
88
89
90     //no one user is following another user yet.
91     //That is, every element in following is false.
92
93 }
94
95 //TODO: find out whether usrn is in the profiles or not.
96 //If yes, return the index, otherwise, return -1.
97 int Network::findID(string usrn) {
98
99
100
101
102
103 }
104
105 //TODO: Attempts to sign up a new user with specified username and displayname
106 // return true if the operation was successful, otherwise return false
107 //The specified usrn needs to contain alphabet and digit letter only. That is, only 'A'-'Z', 'a'-'z',
108     or '0'-'9' are allowed.
109 //Hint: use int isalpha ( int c ) to test
110 //whether character c is alphabetic, ie, 'a'-'z', or 'A'-'Z'.
111 //Similarly, use int isdigit ( int c ) to test
112 //whether character c is decimal digit '0' - '9'.

```



```

112 //To use isalpha or isdigit ,
113 //need to include ctype library.
114 //include <ctype>
115 bool Network::addUser(string usrn, string dspn) {
116     //TODO: if any character in usrn is
117     //neither alphabet nor digit, return false
118
119
120     //TODO: if numUsers is larger than equal to MAX_USERS,
121     //return false.
122
123
124     //TODO: if usrn is not an existing username,
125     //ie, usrn not found in profiles,
126     //add usr to profiles, increase numUsers by one.
127     //
128     //hints: call findID method of Network class with usrn.
129     //If the return is -1, then usrn is not found in profiles.
130     //otherwise, put usrn, dspn profile object to
131     //data member profiles,
132     //increase number of users, denoted by numUsers, by 1.
133     //Do not forget to return true.
134     if (findID(usrn) == -1) { //usrn not found in profiles
135
136
137
138     }
139     else //usrn is an existing user in profiles.
140         //TODO: what is the return?
141
142 }
143
144 //Find out whether usrn1 follows usrn2.
145 //Hint: use findID to see whether usrn1 or usrn2 exists or not.
146 //Put the result of findID on usrn1 to id1,
147 //and the result of findID on usrn2 to id2,
148 //if both usrn1 and usrn2 exist,
149 //ie, neither id1 nor id2 is -1,
150 //set following[id1][id2] to be true and then return true;
151 //otherwise, at least one of id1 and id2 is -1,
152 //set following[id1][id2] to be false and
153 //return false.
154 bool Network::follow(string usrn1, string usrn2) {
155
156
157
158 }
159
160 //TODO: Print Dot file (graphical representation of the network).
161 //Hints:
162 //(1) To print double quotes " symbol,
163 //    use cout << "\"";
164 //(2) Use a nested loop to find the followers of each user.
165 void Network::printDot() {
166

```

```

167 }
168
169 int main() {
170     Network nw;
171     // add three users
172     nw.addUser("mario", "Mario");
173     nw.addUser("luigi", "Luigi");
174     nw.addUser("yoshi", "Yoshi");
175
176     // make them follow each other
177     nw.follow("mario", "luigi");
178     nw.follow("mario", "yoshi");
179     nw.follow("luigi", "mario");
180     nw.follow("luigi", "yoshi");
181     nw.follow("yoshi", "mario");
182     nw.follow("yoshi", "luigi");
183
184     // add a user who does not follow others
185     nw.addUser("wario", "Wario");
186
187     // add clone users who follow @mario
188     for(int i = 2; i < 6; i++) {
189         string usrn = "mario" + to_string(i);
190         string dspn = "Mario " + to_string(i);
191         nw.addUser(usrn, dspn);
192         nw.follow(usrn, "mario");
193     }
194     // additionally, make @mario2 follow @luigi
195     nw.follow("mario2", "luigi");
196
197     nw.printDot();
198 }

```

Sample output is as follows.

```

digraph {
"@mario"
"@luigi"
"@yoshi"
"@wario"
"@mario2"
"@mario3"
"@mario4"
"@mario5"
"@mario"->"@luigi"
"@mario"->"@yoshi"
"@luigi"->"@mario"
"@luigi"->"@yoshi"
"@yoshi"->"@mario"
"@yoshi"->"@luigi"
"@mario2"->"@mario"
"@mario2"->"@luigi"
"@mario3"->"@mario"

```

```

"@mario4"->"@mario"
"@mario5"->"@mario"
}

```

4 Task D

In additional to codes in Task C, do the following:

- Define struct Point. A struct is a simplified class with all data members are public and no methods operating on those data members.
- Define methods writePost and printTimeline for class Network.

```

1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 class Profile {
6 private:
7     string username;
8     string displayname;
9 public:
10    // Profile constructor for a user (initializing
11    // private variables username=usrn, displayname=dspn)
12    Profile(string usrn, string dspn);
13
14    // Default Profile constructor (username="", displayname="")
15    Profile();
16
17    // Return username
18    string getUsername();
19
20    // Return name in the format: "displayname (@username)"
21    string getFullName();
22
23    // Change display name
24    void setDisplayName(string dspn);
25 };
26
27 Profile::Profile(string usrn, string dspn) {
28
29 }
30
31 //TODO: Default Profile constructor (username="", displayname="")
32 Profile::Profile() : Profile("", "") {
33 }
34
35 //TODO: Return username
36 string Profile::getUsername() {
37
38 }
39
40 //TODO: Return name in the format: "displayname (@username)"
41 string Profile::getFullName() {

```

```

42 }
43 }
44
45 //TODO: Change display name
46 void Profile::setDisplayname(string dspn) {
47
48 }
49
50 //ADDED in Task D
51 struct Post{
52     string username;
53     string message;
54 };
55
56 class Network {
57 private:
58     static const int MAX_USERS = 20; // max number of user profiles
59     int numUsers; // number of registered users
60     Profile profiles[MAX_USERS]; // user profiles array:
61                                     // mapping integer ID -> Profile
62
63     // Returns user ID (index in the 'profiles' array) by their username
64     // (or -1 if username is not found)
65     int findID (string usrn);
66
67     bool following[MAX_USERS][MAX_USERS]; // friendship matrix:
68     // following[id1][id2] == true when id1 is following id2
69
70     static const int MAX_POSTS = 100;
71     int numPosts; // number of posts
72     Post posts[MAX_POSTS]; // array of all posts
73
74 public:
75     // Constructor, makes an empty network (numUsers = 0)
76     Network();
77
78     // Attempts to sign up a new user with specified username and displayname
79     // return true if the operation was successful, otherwise return false
80     bool addUser(string usrn, string dspn);
81
82     // Make 'usrn1' follow 'usrn2' (if both usernames are in the network).
83     // return true if success (if both usernames exist), otherwise return false
84     bool follow(string usrn1, string usrn2);
85
86     // Print Dot file (graphical representation of the network)
87     void printDot();
88     bool writePost(string usrn, string msg); // new
89     bool printTimeline(string usrn); // new
90 };
91
92 Network::Network() {
93     //Keep the code in Task C.
94
95     //TODO: set numPosts to be zero.
96     //needed, or numPosts is not initialized

```

```

97 }
98
99 //Same as the code in Task C.
100 int Network::findID(string usrn) {
101
102 }
103
104 //Same as the code in Task C.
105 bool Network::addUser(string usrn, string dspn) {
106
107 }
108
109 //Same as the code in Task C.
110 bool Network::follow(string usrn1, string usrn2) {
111
112 }
113
114 //Same as the code in Task C.
115 void Network::printDot() {
116
117 }
118
119 //TODO: add in Task D.
120 bool Network::writePost(string usrn, string msg) {
121     //if numPosts is larger than or equal to MAX_POSTS,
122     //return false.
123     //Data member numPosts is the index
124     //of next post to write,
125     //it cannot be equal to MAX_POSTS,
126     //let alone to be larger than MAX_POSTS.
127
128
129     //Save usrn and msg to the corresponding attribute of
130     //the (numPosts)th element of posts.
131
132
133     //Increase numPosts by 1.
134
135     //Now a post with info of usrn and msg is
136     //successfully put to array posts,
137     //what should we return?
138
139 }
140
141 //TODO: newly added method of Task D.
142 bool Network::printTimeline(string usrn) {
143     //Key steps:
144     //(1) Check that usrn exists in profiles using findID,
145     // if not, return false.
146     // Put the return in variable id.
147     //(2) Display posts in reverse-chronological order.
148     // Hints: read hints for Task D of Lab 3 in
149     //https://tong-yee.github.io/135/lab_hints/hints_lab3_f23.pdf.
150     //(3) Display all the posts of usrn or his/her followers,
151     // check whether username of the current post is a follower

```

```

152 // of usrn or not. Find id of username and put in id2.
153 // Check following[id][id2] is true or not.
154
155 }
156
157 int main() {
158     Network nw;
159     // add three users
160     nw.addUser("mario", "Mario");
161     nw.addUser("luigi", "Luigi");
162     nw.addUser("yoshi", "Yoshi");
163
164     nw.follow("mario", "luigi");
165     nw.follow("luigi", "mario");
166     nw.follow("luigi", "yoshi");
167     nw.follow("yoshi", "mario");
168
169     // write some posts
170     nw.writePost("mario", "It's a-me, Mario!");
171     nw.writePost("luigi", "Hey hey!");
172     nw.writePost("mario", "Hi Luigi!");
173     nw.writePost("yoshi", "Test 1");
174     nw.writePost("yoshi", "Test 2");
175     nw.writePost("luigi", "I just hope this crazy plan of yours works!");
176     nw.writePost("mario", "My crazy plans always work!");
177     nw.writePost("yoshi", "Test 3");
178     nw.writePost("yoshi", "Test 4");
179     nw.writePost("yoshi", "Test 5");
180
181     cout << endl;
182     cout << "==== Mario's timeline =====" << endl;
183     nw.printTimeline("mario");
184     cout << endl;
185
186     cout << "==== Yoshi's timeline =====" << endl;
187     nw.printTimeline("yoshi");
188     cout << endl;
189 }

```

Sample output is as follows.

```

===== Mario's timeline =====
Mario (@mario) My crazy plans always work!
Luigi (@luigi) I just hope this crazy plan of yours works!
Mario (@mario) Hi Luigi!
Luigi (@luigi) Hey hey!
Mario (@mario) It's a-me, Mario!

===== Yoshi's timeline =====
Yoshi (@yoshi) Test 5
Yoshi (@yoshi) Test 4
Yoshi (@yoshi) Test 3
Mario (@mario) My crazy plans always work!
Yoshi (@yoshi) Test 2

```

Yoshi (@yoshi) Test 1
Mario (@mario) Hi Luigi!
Mario (@mario) It's a-me, Mario!