```cpp
#include <iostream>
#include <map>
#include <queue>
#include <set>
#include <stack>
using namespace std;

//Problem #0
void removeEvens(stack<int> &numStack) {
    stack<int> temp;

    while(!numStack.empty()) {
        if(numStack.top() % 2 != 0)
            temp.push(numStack.top());
        numStack.pop();
    }
    while(!temp.empty()) {
        numStack.push(temp.top());
        temp.pop();
    }
}
void printStackInt(stack<int> theStack) {
    while(!theStack.empty()) {
        cout << theStack.top() << ", ";
        theStack.pop();
    }
}

//Problem #1
bool allUniqueValues(queue<string> theQueue) {
    if(theQueue.empty() || theQueue.size() == 1)
        return true;

    set<string> theSet;

    while(!theQueue.empty()) {
        if(theSet.contains(theQueue.front()))
            return false;
        theSet.insert(theQueue.front());
        theQueue.pop();
    }
    return true;
}

//Problem #2
map<string, set<string>> getCorrectClubMap(map<string, set<string>> &
origMap) {
    map<string, set<string>> newMap;

    for(auto iter = origMap.begin(); iter != origMap.end(); iter++) {
```

```
50          string club = iter->first;
51          set<string> people = iter->second;
52
53          for(string person : people) {
54              newMap[person].insert(club);
55          }
56      }
57      return newMap;
58  }
59
60  void printMapString(map<string, set<string>> clubMap) {
61      for(auto iter = clubMap.begin(); iter != clubMap.end(); iter++) {
62          cout << iter->first << ": ";
63          set<string> clubMembers = iter->second;
64          for(string name : clubMembers)
65              cout << name << ", ";
66          cout << endl;
67      }
68  }
69
70  int main()
71  {
72      //TODO: CHECK IF ANYTHING IS EMPTY
73
74      cout << "Problem #0" << endl;
75      stack<int> theStack;
76      theStack.push(1);
77      theStack.push(2);
78      theStack.push(3);
79      theStack.push(4);
80      theStack.push(5);
81
82      removeEvens(theStack);
83      printStackInt(theStack);
84      cout << endl;
85
86      cout << "Problem #1" <<endl;
87      queue<string> theQueue;
88      theQueue.push("a");
89      theQueue.push("b");
90      theQueue.push("c");
91      theQueue.push("d");
92
93      if(allUniqueValues(theQueue))
94          cout << "All unique values" << endl;
95      else
96          cout << "Not all unique values" << endl;
97
98      cout << "Problem #2" << endl;
99      map<string, set<string>> ogClubMap;
```

```
100
101     ogClubMap["A"] = {"Peyton", "Kelly", "Katherine"};
102     ogClubMap["B"] = {"Sophie", "Peyton"};
103     ogClubMap["C"] = {"Sophie", "Peyton", "Kelly"};
104
105     cout << "Original Map: " << endl;
106     printMapString(ogClubMap);
107
108     map<string, set<string>> newClubMap = getCorrectClubMap(ogClubMap);
109
110     cout << "New Map: " << endl;
111     printMapString(newClubMap);
112
113     cout << "Original Map: " << endl;
114     printMapString(ogClubMap);
115
116
117     // cout << "Problem #0.0: " << endl;
118     // map<string, set<string>> clubMap;
119     // clubMap["A"] = {"Nick", "Kip", "Sammy"};
120     // clubMap["B"] = {"Nick", "Sammy"};
121     // clubMap["C"] = {"Kip", "Sammy"};
122     // printMapString(clubMap);
123     // cout << "Adding Neal..." << endl;
124     // joinClubs(clubMap, "Neal", "Nick");
125     // printMapString(clubMap);
126     //
127     // cout << "Problem #0.1: " << endl;
128     // stack<queue<int>> s;
129     // queue<int> q1;
130     // q1.push(1);
131     // q1.push(2);
132     // q1.push(3);
133     // q1.push(4);
134     // queue<int> q2;
135     // q2.push(2);
136     // q2.push(3);
137     // q2.push(4);
138     // q2.push(5);
139     // queue<int> q3;
140     // q3.push(3);
141     // q3.push(4);
142     // q3.push(5);
143     // q3.push(6);
144     // s.push(q1);
145     // s.push(q2);
146     // s.push(q3);
147     //
148     //
149     // removeFirstElements(s);
```

```
150     // //printStackQueue(s);
151     //
152     // cout << "Problem #0.2" << endl;
153     // map<string, set<string>> travelMap;
154     // travelMap["Neal"] = {"A", "B", "C", "D", "E"};
155     // travelMap["Sammy"] = {"F", "B", "D", "K", "J"};
156     // travelMap["KMo"] = {"F", "B", "A", "D", "L"};
157     // travelMap["Kip"] = {"F", "B", "V", "D", "M"};
158     //
159     // printSetString(getMostVisitedCountries(travelMap));
160     //
161     // cout << "Problem #1.0" << endl;
162     // stack<int> theStack;
163     // theStack.push(1);
164     // theStack.push(1);
165     // theStack.push(5);
166     // theStack.push(2);
167     // theStack.push(1);
168     //
169     // removeAllBottomValues(theStack);
170     // printStackInt(theStack);
171     //
172     // cout<< "Problem #1.1" << endl;
173     // set<queue<string>> setOfQueues;
174     // queue<string> qa;
175     // qa.push("E");
176     // qa.push("B");
177     // queue<string> qb;
178     // qb.push("A");
179     // qb.push("B");
180     // qb.push("C");
181     // qb.push("N");
182     // setOfQueues.insert(qa);
183     // setOfQueues.insert(qb);
184     //
185     // if(uniqueFronts(setOfQueues))
186     //     cout << "Unique" << endl;
187     // else
188     //     cout << "Not unique" << endl;
189
190     return 0;
191 }
192 // void joinClubs(map<string, set<string>> &clubMap, string
    lonelyStudent, string friendlyStudent) {
193 //     for(auto iter = clubMap.begin(); iter != clubMap.end(); iter++) {
194 //         string club = iter->first;
195 //         set<string> clubMembers = iter->second;
196 //
197 //         if(clubMembers.contains(friendlyStudent)){
198 //             clubMap.at(club).insert(lonelyStudent);
```

```
199 //         }
200 //     }
201 // }
202
203
204 //
205 // void removeFirstElements(stack<queue<int>> &s) {
206 //     stack<queue<int>> tempS;
207 //     while(!s.empty()){
208 //         queue<int> q = s.top();
209 //         s.pop();
210 //         q.pop();
211 //         tempS.push(q);
212 //     }
213 //     while(!tempS.empty()){
214 //         s.push(tempS.top());
215 //         tempS.pop();
216 //     }
217 // }
218 //
219 // void printStackQueue(const stack<queue<int>> &s) {
220 //     stack<queue<int>> tempS = s;
221 //     int i = 0;
222 //     while(!s.empty()) {
223 //         queue<int> q = tempS.top();
224 //         tempS.pop();
225 //         cout << i << ": ";
226 //         while(!q.empty()) {
227 //             cout << q.front() << ", ";
228 //             q.pop();
229 //         }
230 //         i++;
231 //         cout << endl;
232 //     }
233 // }
234 //
235 // set<string> getMostVisitedCountries(map<string, set<string>> &
    travelMap) {
236 //     set<string> result;
237 //     map<string, int> freqMap;
238 //     for(auto iter = travelMap.begin(); iter != travelMap.end(); iter
    ++) {
239 //         set<string> countries = iter->second;
240 //         for(string c : countries) {
241 //             freqMap[c]++;
242 //         }
243 //     }
244 //
245 //     int champ = 0;
246 //     for(auto iter = freqMap.begin(); iter != freqMap.end(); iter++) {
```

```
247 //         string country = iter->first;
248 //         int num = iter->second;
249 //
250 //         if(num > champ) {
251 //             champ = num;
252 //             result.clear();
253 //             result.insert(country);
254 //         }
255 //         else if(num == champ)
256 //             result.insert(country);
257 //     }
258 //     return result;
259 // }
260 //
261 // void printSetString(set<string> countries) {
262 //     for(string c : countries) {
263 //         cout << c << ", ";
264 //     }
265 //     cout << endl;
266 // }
267 //
268 // void removeAllBottomValues(stack<int> &theStack) {
269 //     if(theStack.empty())
270 //         return;
271 //     stack<int> temp;
272 //     while(!theStack.empty()) {
273 //         temp.push(theStack.top());
274 //         theStack.pop();
275 //     }
276 //     int badNum = temp.top();
277 //     while(!temp.empty()) {
278 //         if(temp.top() != badNum)
279 //             theStack.push(temp.top());
280 //         temp.pop();
281 //     }
282 // }
283 //
284 // void printStackInt(stack<int> theStack) {
285 //     while(!theStack.empty()) {
286 //         cout << theStack.top() << ", ";
287 //         theStack.pop();
288 //     }
289 // }
290 //
291 // bool uniqueFronts(set<queue<string>> &setOfQueues) {
292 //     set<string> fronts;
293 //     if(setOfQueues.size() < 2)
294 //         return true;
295 //     for(queue<string> q : setOfQueues) {
296 //         if(!q.empty()){
```

```
297 //             if(fronts.count(q.front()) > 0)
298 //                 return false;
299 //             fronts.insert(q.front());
300 //         }
301 //     }
302 //     return true;
303 // }
```