题目描述：

推荐多样性需要从多个列表中选择元素，一次性要返回 N 屏数据（窗口数量），每屏展示 K 个元素（窗口大小），选择策略：

1. 各个列表元素需要做穿插处理，即先从第一个列表中为每屏选择一个元素，再从第二个列表中为每屏选择一个元素，依次类推

2. 每个列表的元素尽量均分为 N 份，如果不够 N 个，也要全部分配完，参考样例图：

（1）从第一个列表中选择 4 条 0 1 2 3，分别放到 4 个窗口中

（2）从第二个列表中选择 4 条 10 11 12 13，分别放到 4 个窗口中

（3）从第三个列表中选择 4 条 20 21 22 23，分别放到 4 个窗口中

（4）再从第一个列表中选择 4 条 4 5 6 7，分别放到 4 个窗口中

...

（5）再从第一个列表中选择，由于数量不足 4 条，取剩下的 2 条，放到窗口 1 和窗口 2

（6）再从第二个列表中选择，由于数量不足 4 条并且总的元素数达到窗口要求，取 18 19 放到窗口 3 和窗口 4

输入

N：4
K：7

recallA	0	1	2	3	4	5	6	7	8	9
recallB	10	11	12	13	14	15	16	17	18	19
recallC	20	21	22	23	24	25	26	27	28	29

处理

window1	window2	window3	window4
0	1	2	3
10	11	12	13
20	21	22	23
4	5	6	7
14	15	16	17
24	25	26	27
8	9	18	19

输出

0 10 20 4 14 24 8 1 11 21 5 15 25 9 2 12 22 6 16 26 18 3 13 23 7 17 27 19

window1 window2 window3 window4

输入描述：

第一行输入为 N，表示需要输出的窗口数量，取值范围 [1, 10]

第二行输入为 K，表示每个窗口需要的元素数量，取值范围 [1, 100]

之后的行数不定（行数取值范围[1,10]），表示每个列表输出的元素列表。元素之间以空格分隔，已经过排序处理，每个列表输出的元素数量取值范围[1,100]

输出描述：

输出元素列表，元素数量=窗口数量*窗口大小，元素之间以空格分隔，多个窗口合并为一个列表输出，参考样例：

先输出窗口 1 的元素列表，再输出窗口 2 的元素列表，再输出窗口 3 的元素列表，最后输出窗口 4 的元素列表

补充说明：

1. 每个列表会保证元素数量满足窗口要求，不需要考虑元素不足情况

2. 每个列表的元素已去重，不需要考虑元素重复情况

3. 每个列表的元素列表均不为空，不需要考虑列表为空情况

4. 每个列表的元素列表已经过排序处理，输出结果要保证不改变同一个列表的元素顺序

5. 每个列表的元素数量可能是不同的

示例 1

输入：

```
4
7
0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
```

输出：

```
0 10 20 4 14 24 8 1 11 21 5 15 25 9 2 12 22 6 16 26 18 3 13 23 7 17
27 19
```

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <unordered_map>
#include <unordered_set>
#include <set>
#include <queue>
#include <map>
#include <deque>
#include <sstream>
#include <cstring>
#include <functional>

using namespace std;

vector<string> split(string s, char c) {
    vector<string> v;
    int l = 0, r = 0, n = s.size();
    while (r < n) {
        while (s[l] == c && l < n) {
            l++;
            r++;
        }
        if (l == n) break;
        while (s[r] != c && r < n) r++;
        v.push_back(s.substr(l, r - l));
        l = r + 1;
        r = l;
    }
    return v;
}

int main()
{
    int windowNum, windowSize;
    cin >> windowNum >> windowSize;
    // 如果前面有输入，则清除
    cin.ignore(100, '\n');
    string line;
    deque<deque<int>> Lists;
    while (getline(cin, line)) {
```

```
        if (line.empty()) break;

        vector<string> v = split(line, ' ');
        deque<int> list;
        for (int i = 0; i < (int)v.size(); i++) {
                list.push_back(stoi(v[i]));
        }
        Lists.push_back(list);
}

// for(int i = 0; i < (int)Lists.size(); i++) {
//         for(int j = 0; j < (int)Lists[i].size(); j++) {
//                 cout << Lists[i][j] << " ";
//         }
//         cout << endl;
// }

vector<vector<int>> windows(windowNum);

int listIdx = 0;
// int windowIdx = 0;
while (1) {
        int getNum = 0;
        vector<int> getNumList;
        while (getNum < windowNum) {
                while (Lists[listIdx].empty()) {
                        listIdx++;
                        listIdx = listIdx % (int)Lists.size();
                }

                int curNum = Lists[listIdx].front();
                getNumList.push_back(curNum);
                Lists[listIdx].pop_front();
                getNum++;
        }
        listIdx++;
        listIdx = listIdx % (int)Lists.size();

        for (int i = 0; i < (int)getNumList.size(); i++) {
                windows[i].push_back(getNumList[i]);
        }
        if (windows[0].size() == windowSize) break;
}
```

```
	for (int i = 0; i < (int)windows.size(); i++) {
		for (int j = 0; j < (int)windows[i].size(); j++) {
			cout << windows[i][j] << " ";
		}
	}

	return 0;
}

/*
4
7
0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29

*/
```