SI 335 Spring 2012: Project 2

This project contains both written and electronic (programming) parts. The electronic parts (code) must be submitted according to the instructions on this page. The written parts must be typed or neatly written, and handed in to the instructor.

Both parts of this project are due before the beginning of class on **Friday, February 10**. Late submissions will not be accepted, in accordance with the class policy. Students with excused absences on that date may email the written portion by the deadline *in addition to handing in a paper copy at the earliest opportunity.*

Coding Guidelines:

- It should be clear what your code is doing. The best way is to use sensible variable/function/class names, and to clearly state what every non-trivial chunk of your code (function, long loop, long block) does.
- Your program should read from standard in and write to standard out. Its I/O behavior should be exactly identical to the sample starter code. *You should write any error or debugging messages to stderr (cerr)*, as this stream will be ignored in testing.
- Follow the course honor policy: document any external sources, and don't produce or look at any code in collaboration with any classmate.

Written Guidelines:

- All work must be typed or neatly written
- Multiple-page submissions must be stapled or otherwise bound securely into a single packet.
- **Be concise**. Most problems have at least one simple and relatively short correct answer. Solutions will be graded not only on their correctness but also on how clearly that solution is presented.
- In general, there is a preference for **correct** algorithms over efficient ones. If the problem asks for a $O(n)$ algorithm, a *correct* $O(n \log n)$ algorithm is usually preferable over an incorrect one that claims $O(n)$ running time.
- **Do not try to pass off a solution you know is incorrect**. A statement to the effect of "I know this doesn't work because of blah" will be effective in garnering partial credit points.

Service Selection at the OMA

The year is 1984 and the world is ruled by just a few super-states. One of them, Oceania, is dominant and controls both North and South America, sourthern Africa, and Australia. It is engaged in a perpetual, unwinnable war with the other super-powers for the purpose of subjugating its own citizens.

To support the perpetual war, the Oceania Military Academy (OMA) churns out an amazingly high number of graduates every year (in the millions), and the process of selecting which part of the military service they will each enter is a daunting task. This being Oceania, the students themselves (called Proles) have no say in their own service selection. Instead, they are all ranked by each of the different communities, who then take turns in picking which Proles they will get.

Particulars

Your task is to design an algorithm and write a computer program that reads in the names of the k different communities, as well as the n Proles and their rankings in each community, and prints the results of the service selection. This works similarly to a draft: the k communities, in the order they were read in, take turns picking, one at a time, until there are no more Proles left. For each pick at each iteration, the community chooses the highest-ranking Prole (according to *its own ranking*) that has not been picked yet.

Specifically, the input, which should be read from standard in, contains in order

- The integer k
- k strings (each without any spaces) for the community names
- The integer n
- n lines, each containing a string (with no spaces) for the Prole's name, followed by k positive integers, for the rankings of that Prole by each community.

Example

For example, consider the following sample input file:

```
3
Tanks
Infantry
Paratrooper
5
Bill 4 5 1
Suzy 1 1 4
John 2 4 5
Kate 3 2 3
Mike 5 3 2
```

The Tanks community will choose Suzy first. Then the Infantry would like to choose Suzy but can't, so they pick Kate instead. Next the Paratroopers pick Bill, and so on. The correct output would be

```
Suzy Tanks
Kate Infantry
Bill Paratrooper
John Tanks
Mike Infantry
```

Provided Code

The following C++ programs are available to help you. You can download them all at once (in a gzipped tarball) from this link. Download the file, then type `tar xzvf proj2.tar.gz` from the linux command line to extract the files into a folder called `proj2`.

- selection.cpp contains an implementation of my algorithm for the problem, described below.
- gencases.cpp is a program to generate test cases for given values of k and n. It relies on a file of names called `random-names.txt`. A huge file with random alien names is provided in the tarball linked above.
- check.cpp is a program to check whether a given input to the problem is valid. This is not trivial, since it has to check whether each ranking appears exactly once, for each community.

Here is a pseudocode description of the algorithm that appears in the selection.cpp file.

Simple OMA service selection algorithm
Input: Integers `k` and `n`, and n k-tuples of rankings. Each ranking, for each community is an integer between 1 and n.
Output: The list of service selections

```
1  Read the rankings into an array
2  for i from 0 to n-1 do
3    r := 1
4    c := i mod k
5    do
6      Find Prole p with rank r by community c using linear search
7    while p is marked
8    mark p
9    print the names of Prole p and community c
10 end for
```

Your Tasks

The first two tasks are written, and the last is electronic. See the instructions at the top of this assignment for how to submit each part.

1. Analyze my "simple service selection algorithm" described above and provided in code. Use both sources (pseudocode and actual code) to help you understand the algortihm. I want (at least) a big-O analysis of the worst-case cost, in terms of k and n. Try to make your analysis as tight as possible.
2. Describe an improved algorithm for the same problem in pseudocode, and give a worst-case big-O analysis, in terms of k and n. Try to give a tight analysis, and try to design your algorithm to be as efficient as possible. If necessary, you may assume that k is much smaller than n. So for example, $O(kn)$ time is much better than $O(n^2)$ time.
3. Implement your improved algorithm, along with any other tweaks you can think of, so that your code is as fast as possible. I recommend that you follow my example and code in C++, but you may use any programming language that you wish, as long as it runs in the linux environment in the labs. In any case, your improved program should be *in a single file* called `sel.EXT`, where EXT is `cpp` or another extension depending on your language. If you don't use C++, you must also submit a `README.txt` file with instructions on how to compile and run your program.

Bonus: The fastest (correct) program on a large sample input of my choosing will receive a 10% bonus on the grade for this project.