CSCI-1200 Data Structures — Fall 2018
Homework 4 — Preference Lists

In this assignment you will write a program to manage the preferences and rankings between schools and applicants who wish to enroll at those schools. Your program will handle several different operations: adding a school and the ranking of students that the school is interested in admitting to their program, adding a student and the ranking of schools that applicant is interested in attending, editing (adding/removing) preferences, and printing the current state of these lists. *Please carefully read the entire assignment before beginning your implementation.*

We provide the code to parse an input file that calls the operations listed above. Furthermore, we provide an implementation of the Gale-Shapley algorithm to compute an assignment of students to schools that is satisfying to both the schools and the students. This famous problem is called the "Stable Marriage Problem", which refers to the original problem statement:

> Given n men and n women, where each person has ranked all members of the opposite sex with a unique number between 1 and n in order of preference, marry the men and women together such that there are no two people of opposite sex who would both rather have each other than their current partners. If there are no such people, all the marriages are "stable".

This algorithm enjoys real-world application each year to the problem of assigning medical residents to hospitals. For more information and discussion about the problem, the algorithm, and the optimality of the solution, see http://en.wikipedia.org/wiki/Stable_marriage_problem.

Input/Output

The input for the program will come from a file and the output will also go to a file. These file names are specified by command-line arguments. Here's an example of how your program will be called:

```
preferences.exe requests.txt results.txt
```

The form of the input is relatively straightforward and easy to read. Each request begins with a special keyword. There are nine different requests, described below. You may assume the input file strictly follows this format (i.e., you don't need to worry about format error-checking).

```
add_school university_of_michigan 3 3
erin_jones
john_smith
dave_roberts

insert_student_into_school_preference_list university_of_michigan joe_miller dave_roberts

print_school_preferences university_of_michigan
```

The examples above illustrate how a school's preference lists are managed. The `add_school` request is followed by a string that represents the school's name (using `_` instead of spaces), an integer that represents the number of available slots in the program, an integer that represents the number of students that are will initially be placed on their preference list, and followed by the names of those students as strings (again using `_` rather than spaces). After initial construction of a school's list, the list may be expanded with the `insert_student_into_school_preference_list` request, which takes the name of the school, the name of the new student to be added, and the name of a student who is already on the list in front of which this new student should be inserted. Finally, the current state of a school's preference list can be printed with the `print_school_preferences` command, which will result in the following text being written to the output file for this example:

```
university_of_michigan preference list:
  1. erin_jones
  2. john_smith
  3. joe_miller
  4. dave_roberts
```

Somewhat similarly, the student preference list is initially constructed with the `add_student` request, which is followed by a string representing the student's name, an integer representing the number of schools in the initial ranking, and the names of those schools. While schools may add students into their preference ranking after initial construction, students have the opposite operation, students may cross off schools in their preference list using the `remove_school_from_student_preference_list` command.

```
add_student erin_jones 3
duke_university
university_of_michigan
university_of_california_san_francisco

remove_school_from_student_preference_list erin_jones university_of_california_san_francisco

print_student_preferences erin_jones
```

The above requests will result in the following text being added to the output file:

```
erin_jones preference list:
  1. duke_university
  2. university_of_michigan
```

The final set of requests pertains to the matching of students to schools using the Gale-Shapley algorithm:

```
perform_matching
print_school_enrollments
print_student_decisions
```

The matching algorithm proceeds through a series of rounds or iterations. In each round, each school that has open slots in its program, makes an offer to the next student on their preference list. When a student receives an offer, he/she compares this new offer to his/her previous best offer (if any), and if this offer is better, tentatively accepts this new offer (and declines the previous offer). Otherwise, the offer is declined. The algorithm terminates after a round in which no offers are made (because all schools have either filled their slots *or* schools with openings have exhausted their preference list of applicants). The progression of the algorithm can be monitored by examining a log of these offers:

```
---- ROUND 1 ----
university_of_michigan makes an offer to erin_jones
  erin_jones tentatively accepts offer from university_of_michigan
duke_university makes an offer to john_smith
  john_smith declines offer from duke_university
university_of_california_san_francisco makes an offer to erin_jones
  erin_jones declines offer from university_of_california_san_francisco
---- ROUND 2 ----
university_of_michigan makes an offer to john_smith
  john_smith tentatively accepts offer from university_of_michigan
duke_university makes an offer to erin_jones
  erin_jones withdraws tentative acceptance of offer from university_of_michigan
  erin_jones tentatively accepts offer from duke_university
university_of_california_san_francisco makes an offer to joe_miller
  joe_miller declines offer from university_of_california_san_francisco
```

```
---- ROUND 3 ----
university_of_michigan makes an offer to joe_miller
  joe_miller declines offer from university_of_michigan
university_of_california_san_francisco makes an offer to john_smith
  john_smith declines offer from university_of_california_san_francisco
---- ROUND 4 ----
university_of_michigan makes an offer to dave_roberts
  dave_roberts tentatively accepts offer from university_of_michigan
---- ROUND 5 ----
no offers_made this round
```

The full solution can be printed in two formats: by school or by student (both shown below). The enrollments for each school are printed ordered alphabetically by school and then alphabetically by student. The student decisions are printed ordered alphabetically by student. Note that for simplicity, we treat the name of each school and each student as a single string (using _ rather than spaces), and define "alphabetically" for these names as the standard simple ordering of strings.

```
student(s) who will be attending duke_university:
  erin_jones
student(s) who will be attending university_of_california_san_francisco:
  [2 remaining slot(s) in enrollment]
student(s) who will be attending university_of_michigan:
  dave_roberts
  john_smith
  [1 remaining slot(s) in enrollment]

dave_roberts will be attending university_of_michigan
erin_jones will be attending duke_university
joe_miller has not received an acceptable offer
john_smith will be attending university_of_michigan
```

Sample input and output files are posted on the course web site. Please follow these examples exactly. In particular note the error messages that are printed for attempted duplicate insertion or removal of elements that are not in the preference list.

Assignment Requirements, Hints, and Suggestions

- **You may not use vectors or arrays for this assignment.** Use STL lists instead. You may not use maps, or sets, or things we haven't seen in class yet. Be sure to use of const and pass/return by reference where appropriate (refer to the diagram in Lecture 9).
- We have provided a partial implementation of this program, focusing on input parsing and the implementation of the Gale-Shapley algorithm. There are member function calls to our versions of the `School` and `Student` classes, so you can deduce how some of the member functions in our solution work. You are *strongly encouraged* to examine this code carefully and follow the interfaces suggested without modification to the provided code.
- In your README.txt file, provide a Big O Notation complexity analysis of each operation assuming m schools, s slots per school, r rankings of students by each school, n students, and p preferences of schools by each student.
- For extra credit, once your program is working perfectly, select two different simple iterative loops (using `for` or `while`) in your code and rewrite those loops to instead use recursion. Comment out and clearly label the original iterative version of the code and leave it next to the new recursive version. *Note: Normally you should not leave large portions of early draft work in your homework submissions!*
- Be sure to make up new test cases to fully test your program. Use the template `README.txt` to list your collaborators, your time spent on the assignment, and any notes you want the grader to read.