COP 3502C Programming Assignment # 3

Recursion

Read all the pages before starting to write your code

Please include the following comments at the beginning of your code to **declare your authorship of the code:**

```
/* COP 3502C Assignment 3
This program is written by: Your Full Name */
```

Compliance with Rules: UCF Golden rules apply towards this assignment and submission. Assignment rules mentioned in syllabus, are also applied in this submission. The TA and Instructor can call any students for explaining any part of the code in order to better assess your authorship and for further clarification if needed.

Caution!!!

Sharing this assignment description (fully or partly) as well as your code (fully or partly) to anyone/anywhere is a violation of the policy. I may report to office of student conduct and an investigation can easily trace the student who shared/posted it. Also, getting a part of code from anywhere will be considered as cheating.

Deadline:

See the deadline in Webcourses. After that the assignment submission will be locked. An assignment submitted by email will not be graded and such emails will not be replied according to the course policy.

What to do if you need clarification on the problem?

I will create a discussion thread in webcourses and I highly encourage you to ask your question in the discussion board. Maybe many students might have same question like you. Also, other students can reply and you might get your answer faster. Also, you can write an email to the TAs and put the course teacher in the cc for clarification on the requirements.

How to get help if you are stuck?

According to the course policy, all the helps should be taken during office hours. Occasionally, we might reply in email.

PA3: Where to Sit?

Objective

Give practice with recursion.
Give practice with functions in C.
Give practice with creating a design for a program without a given list of functions or structs.

Background Story

You and your friends are planning to get together to attend a movie! However, there are a few restrictions on where everyone can sit:

- Some people don't want to sit next to each other.
- Everyone should have access to popcorn! (This means that for each person, either they bought popcorn, or the person directly to their left or the person directly to their right did.)

For example, imagine that there are five people who want to attend the movie: Alia, Belinda, Carlos, Danica and Edward, where only Alia and Edward buy popcorn. In addition, Alia and Carlos can't sit next to each other and Belinda and Edward can't sit next to each other. Given these restrictions, they can sit in a single row in 10 possible orders.

Alia	Belinda	Carlos	Edward	Danica
Alia	Belinda	Danica	Edward	Carlos
Belinda	Alia	Danica	Carlos	Edward
Belinda	Alia	Danica	Edward	Carlos
Carlos	Edward	Danica	Alia	Belinda
Carlos	Edward	Danica	Belinda	Alia
Danica	Alia	Belinda	Carlos	Edward
Danica	Edward	Carlos	Belinda	Alia
Edward	Carlos	Belinda	Alia	Danica
Edward	Carlos	Danica	Alia	Belinda

Problem

Write two related programs where, given the list of people who are going to the movies together, the pairs of people who can't sit next to each other, and the list of people who are buying popcorn, determines the two following things (Note that these programs will not exchange any data between them. They are just independent code and run independently. Both of them will receive same inputs)):

(1) Program 1 – the number of different orderings (permutations) of the movie attendees that satisfy all the restrictions.

(2) Program 2 – the first ordering (in lexicographical order) of the movie attendees that satisfy all the restrictions.

Input

The first line of input contains two positive integers: n ($3 \leq n \leq 10$), the number of people attending the movie, and p ($0 \leq p \leq n$), the number of pairs of people who do not want to sit next to each other.

The next n lines will contain the information about each of the people attending the movie, with one person described per line. These lines will describe the people in alphabetical order. Each of these lines will have the following format:

```
NAME 0/1
```

Each name will be an uppercase alphabetic string with no more than 19 characters. If the number 0 is on the line, this indicates that that person does not have popcorn. If the number 1 is on the line, this indicates that that person does have popcorn.

The following p lines will each contain a pair of names, indicating two people who do not want to sit next to each other. It is guaranteed that each of the $2p$ names appearing in this section will be one of the n names listed previously as the movie attendees. Secondly, it's guaranteed that the two names on a single line will be distinct.

Output (for Program 1)

On a single line, simply output the total number of valid orderings of the people attending the movie sitting together in a single row, from left to right. It is guaranteed that the input data will be such that this value will be a positive integer.

Output (for Program 2)

Output, with one name per line, the first lexicographical valid ordering of the people attending the moving sitting together in a single row, from left to right. In lexicographical ordering, all lists starting with name1 will come before all lists starting with name2 if name1 comes before name2 alphabetically. Specifically, given two lists, to determine which one comes first lexicographically, find the first corresponding name on both lists that don't match. Which ever name comes first alphabetically, is the list that comes first in lexicographical order. **(Hint: since the given names are already in alphabetical order, the permutation algorithm shown in class will naturally evaluate the permutations in lexicographical order. Thus, to solve this program, the first valid solution found while running the algorithm will be the correct answer.)**

Sample Input	Sample Output 1	Sample Output 2
5 2 ALIA 1 BELINDA 0 CARLOS 0 DANICA 0 EDWARD 1 ALIA CARLOS BELINDA EDWARD	10	ALIA BELINDA CARLOS EDWARD DANICA
8 3 ALEX 1 ELLIE 1 FRANKLYN 0 JAMELLE 0 MARTY 1 PRI 1 SAMANTHA 1 WES 0 ALEX WES ELLIE MARTY ELLIE WES	10248	ALEX ELLIE FRANKLYN JAMELLE MARTY PRI SAMANTHA WES
6 5 ANEESHA 0 ARTHUR 0 JACQUELINE 1 MATTHEW 1 MEGAN 0 ROBINSON 0 ROBINSON ARTHUR MATTHEW MEGAN MATTHEW ARTHUR ROBINSON MEGAN MEGAN ANEESHA	2	MEGAN JACQUELINE ARTHUR ANEESHA MATTHEW ROBINSON

Sample Explanation

In each of the three cases, we can verify that the outputted list is valid. (In the first sample, Alia and Edward have popcorn and are in seats 1 and 4. Everyone else is adjacent to one of these two seats. In addition both Alia and Carlos are separated and Belinda and Edward are separated.)

In the second sample case, many orderings are possible simply because most of the attendees have popcorn and there are limited pairs of attendees who can't sit next to each other.

In the last sample, popcorn must be in seats 2 and 5. The five pairs of constraints reduces the possibilities to what is listed and its reverse ordering.

Implementation Requirements/Run Time Requirements

1. No dynamically allocated memory is necessary. All the memory requirements are relatively small.

2. The permutation algorithm from class must be used (The **used** helper array approach). The run time of each program should be roughly O(n x n!). (n! for each permutation and n for evaluating if a permutation is a valid arrangement of the movie attendees.)

3. You **may use** global variables to clean up your code so it's easier to read. Please use them sparingly. (Here are the ones recommended: number of people attending the movie, list of names, the list of who has popcorn, and a two dimensional array storing who is allowed to sit next to whom.)

4. Your code must compile and execute on the Codegrade system.

Deliverables

1. Please submit a source file, `main1.c` for your solution to problem A (where the output is a single number).

2. Please submit a source file, `main2.c`, via Webcourses, for your solution to problem B (where the output is a list of names).

Some more Hints:

- Use the permutation code discussed in the class that uses the **used** helper array (not the swapping one) [Check the permutation pdf]
- We have/will discuss(ed) in the very last part of the lecture on how to use that same algorithm to perform permutation any list of objects (in the lecture of permutation with recursion)
- Probably, write 3 more functions one for checking popcorn matters, one for order of seating without conflict, and one that will get status from both of these functions and decide if a given permutation is valid or not
-