Computer Science I Program 3: Where to Sit? (Recursion)

Please Check Webcourses for the Due Date

Please read the whole assignment before you start coding

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:

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

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

* See appendix for details.

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: The given names are already in alphabetical order. There are permutation algorithms on webcourses that 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 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. The C compiler will be gcc using `-std=gnu11` as the standard.

Deliverables

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

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

Appendix

Lexicographically Earlier Strings - A string (S) is lexicographically earlier than a different string (T), if at the first index at which the two strings differ the string S has a character with a lower ASCII value than the string T.

Examples

"Abba" is earlier than "B"
"B" is earlier than "abba"
"candle" is earlier than "cat"
"dear" is earlier than "dearly"

Lexicographically Earlier Permutation - A permutation (P) is lexicographically earlier than a different permutation (Q), if at the first location at which they differ P contains a lexicographically earlier value. For a permutation of strings, this means that the String of P should be lexicographically earlier than the String of Q.

Examples

"Alice", "Carol", "Bob" is earlier than "Bob", "Alice", "Carol"
"Dave", "Carol", "Alice", "Bob" is earlier than "Dave", "Carol", "Bob", "Alice"
"Bob", "bob", "David", "Alice" is earlier than "bob", "Alice", "Bob", "David"