Computer Based Test

1. *For each question, there will be a testcase (e.g. the testcase for question 1 is* testcase1.txt) *Please note that the testcase you received may not necessarily to be the same as the testcase used for grading. Please consider edge cases.*
2. *Let your program read the corresponding testcase and print the result to a corresponding text file(e.g. the output for question 2 should be* output2.txt*).*
3. *For each question, we have provided a corresponding answer (e.g assume you're solving problem 3, the corresponding answer is* ans3.txt*), and your output should be same as answer.*
4. *For each question, your program file should be named* DS{problem number}_{student ID}.c. *(e.g. assume your student ID is* 7112056067 *and you're solving question 3, your program file should be* DS3_7112056067.c)
5. *Each question is scored independently, but partial credit is not awarded; full credit is given only for complete correctness.*
6. *You can submit for three times.*

1 Problem 1

Many programming languages do not support extremely large integers by default. For instance, the value of $100!$ (100 factorial) already exceeds the range of standard types like `int`, `long`, or even `unsigned long long`. In such cases, we must simulate big number arithmetic ourselves.

In this problem, your task is to compute the factorial of a number n using a singly linked list to represent large numbers. Each digit of the number should be stored in a single node, and all arithmetic (especially multiplication) must be performed manually, digit by digit, as taught in elementary school.

Rules

- You must use your own implementation of big number arithmetic using linked lists.
- The input $1 \leq n \leq 50$
- A recommended structure for storing digits is as follows:

```
typedef struct Node {
    int digit;
    struct Node* next;
} Node;
```

Sample Input

```
5
10
20
```

Sample Output

```
5! = 120
10! = 3628800
20! = 2432902008176640000
```

2 Problem 2

At a large hospital, patients arrive and line up at the reception to check in. Each patient is associated with a particular registration counter based on their health department (e.g., pediatrics, cardiology, dermatology, etc.). However, to optimize the check-in process, the hospital arranges the overall waiting line in a special way:

Rules

- When a patient arrives:
  - If other patients from the same counter are already waiting, the new patient is inserted directly behind the last of them.
  - If no one from their counter is in the line, they simply join at the very end of the queue.
- Patients are always served from the front of the line.

To simulate this process, you are required to implement a queue system. Specifically, you must implement a double-ended queue (deque) using a linked list to manage the internal queue structure.

- You must use the following structure, or else you will receive a score of 0.

```
struct MemberNode {
    int id;
    MemberNode* next;
};

struct TeamNode {
    int teamID;
    MemberNode* front;
    MemberNode* rear;
    TeamNode* next;
};
```

Input Format

The input consists of multiple test cases. Each test case begins with an integer t ($1 \leq t \leq 1000$), representing the number of reception counters.

Then t lines follow, each line starting with an integer m_i ($1 \leq m_i \leq 1000$), followed by m_i patient IDs (integers in the range $[0, 999999]$), indicating the patients registered under that counter.

After that, a sequence of commands will follow:

- `ARRIVE x` — patient with ID x arrives and joins the line.
- `SERVE` — the next patient in line is served.
- `CLOSE` — end of the current test case.

The entire input ends when a line with a single `0` is encountered.
Each test case may contain up to 200,000 commands.

Output Format

For each test case, output: Case k, where k is the test case number (starting from 1). Then, for every `SERVE` command, output the ID of the patient being served, each on a separate line. Print a blank line after each test case, including the last one.

Sample Input

```
2
3 101 102 103
3 201 202 203
ARRIVE 101
ARRIVE 201
ARRIVE 102
ARRIVE 202
ARRIVE 103
ARRIVE 203
SERVE
SERVE
SERVE
SERVE
SERVE
SERVE
CLOSE
2
5 259001 259002259003 259004 259005
6 260001 260002260003 260004 260005 260006
ARRIVE 259001
ARRIVE 260001
ARRIVE 259002
ARRIVE 259003
ARRIVE 259004
ARRIVE 259005
SERVE
SERVE
ARRIVE 260002
ARRIVE 260003
SERVE
SERVE
SERVE
SERVE
CLOSE
0
```

Sample Output

```
Case #1
```

```
101
102
103
201
202
203

Case #2
259001
259002
259003
259004
259005
256001
```

3 Problem 3

The company's most recent invention is a panel of illuminated buttons in r rows and c columns. The buttons are numbered left-to-right, top-to-bottom, starting at 1 in the upper-left corner. Each button has two states: lit and unlit. Initially, all buttons are unlit. Pressing a button switches the state of some buttons from lit to unlit (or vice-versa) according to a 3×3 pattern. Pressing a button on a panel applies the pattern centered on that button. To unlock the panel, the buttons must be pressed in such a way so as to light all of them.

Rules

- Use DFS, or else you will receive a score of 0.

Input Format

Each input case will begin with the number of rows and columns on the panel, $1 \le r, c \le 5$ alone on a line. The next three lines describe how pressing a button will affect the nearby lights. This description consists of a 3×3 character grid, where the character '*' indicates that the light in that position switches state (from lit to unlit or from unlit to lit) while '.' means its state remains unchanged. Input ends with 0 0 alone on a line.

Output Format

For each input case, output 'Case #' followed by the number of the case. If there is no way to turn on all the lights, print 'Impossible.' If it is possible to turn on the lights, output the buttons to be pressed in increasing order, separated by single space. Output the answer that requires the fewest number of buttons possible to be pressed. If there is more than one correct solution, anyone will do.

Sample Input

```
2 3
**.
.*.
*..
4 5
.*.
***
.*.
2 2
...
.**
...
4 3
*.*
...
..*
```

```
0 0
```

Sample Output

```
Case #1
2 5 6
Case #2
2 3 4 7 9 12 14 17 18 19
Case #3
1 3
Case #4
Impossible.
```