2. Play Onwards

Two years have passed since Seohun became tired of listening to Yonghun's annoying jokes. Today Yonghun devised a new type of joke using antonyms. For example:

> "What is the antonym of 'program'? Beginner-ton! lolololololol"

Being shocked by Yonghun's antonym joke, Seohun wrote all antonym jokes from Yonghun and analyzed them to spot these patterns:

1. Each word consists of lowercase Latin alphabet 'a'-'z'.
2. When Seohun says an M-letter word, Yonghun responds with an M-letter word.
3. When responding, Yonghun picks a word from a dictionary that contains N M-letter words.
4. Among all words in the dictionary, Yonghun picks the most *different* one from the word Seohun said. The *difference* between word A and word B is measured by the number of indices i ($0 \leq i \leq M-1$) where $A[i] \neq B[i]$.
5. If there are several such words, Yonghun picks any one of them and responds with that word.

Now Seohun is such an expert on antonym jokes that he can correctly figure out what he said to Yonghun by just listening to the response of Seohun. You can do it too! Write a program that says something to Yonghun and another program that guesses what you said, given only the response of Yonghun.

Implementation details

You have to submit two files.

The name of the first file is `make.cpp`. It represents your behavior and should implement the following function. The file should include `make.h`.

```
string MakeWord (int N, int M, string[] Dict)
```

- N: the number of words in the dictionary.
- M: length of each word.
- $Dict$: an array of length N. For each i ($0 \leq i \leq N-1$), $Dict[i]$ is an M-letter word contained in the dictionary. All words in the dictionary are distinct.
- This function is called exactly once per test case.
- This function should return a string of length M, which is the word you will say to

Yonghun. This string should only contain lowercase Latin alphabet 'a'-'z'.

The name of the second file is `guess.cpp`. It also represents your behavior and should implement the following function. The file should include `guess.h`.

```
string GuessWord (int M, string YH)
```

- M: the length of the word.
- YH: an array of length M. It is the response of Yonghun after listening to your word.
- The function is called exactly once per test case.
- This function should return a string of length M. It should be equal to the word you said. i.e., the return value of `MakeWord`.

If some of the above conditions are not satisfied, your program is judged as `Wrong Answer`. Otherwise, your program is judged as `Accepted`.

Important Notice

- During the actual grading, these two programs are compiled independently.
- Both time and memory usage are measured by the sum of two processes.
- Your program should not use standard input and standard output. Your program should not communicate with other files by any methods. If you use standard input or standard output, your program may be judged as `Wrong Answer`, but we cannot guarantee what would happen.

Example

Consider the following call.

```
MakeWord(4, 7, ["weekend", "evening", "chicken", "alcohol"])
```

Suppose the return value of `MakeWord` is `"cafeine"`.

Then, the following call is made:

```
GuessWord(7, "alcohol")
```

For your program to be judged as `Accepted`, the return value of `GuessWord` should be `"cafeine"`, which equals to the return value of `MakeWord`.

Constraints

- $1 \leq N \leq 100$
- $1 \leq M \leq 10$
- $|Dict[i]| = M$ (for each $0 \leq i \leq N-1$)
- $Dict[i]$ only consists of lowercase Latin alphabet `a-z`. (for each $0 \leq i \leq N-1$)
- $Dict[i] \neq Dict[j]$ (for each $0 \leq i < j \leq N-1$)

Subtasks

1. (100 points) No additional constraints.

The grading criterion for this task is quite different. Let T be the total number of test cases, and t be the number of test cases where your program is judged as `Accepted`. Then, you get P points, where P is defined as:

- $P = \lfloor 100 \times t/T \rfloor$

Sample grader

You can download the sample grader package on the same page you downloaded the problem statement. (scroll down if you don't see the attachment)

If you use IDEs like Visual Studio, Eclipse or Code::Blocks, then import `make.cpp`, `make.h`, `guess.cpp`, `guess.h` and `grader.cpp` into one project and you will be able to compile all these files at once.

If you want to compile by yourself, refer to the following compilation command:

```
g++-7 -Wall -lm -static -DEVAL -o onwards -O2 grader.cpp make.cpp gue
ss.cpp -std=c++17
```

Note that the actual grader is different from the sample grader. The sample grader will be executed as a single process, which will read input data from the standard input and write the results to the standard output.

You should submit only `make.cpp` and `guess.cpp`.

Input format

- line 1: N M
- line $2+i$ (for each $0 \leq i \leq N-1$): $Dict[i]$

Output format

If your program is judged as `Accepted`, the sample grader prints `Correct` in the first line.

If your program is judged as `Wrong Answer`, the sample grader prints the error message in the first line.