COVID tests

In Adam's school, there has been a new wave of the COVID epidemic. In order to prevent further spreading, the school has decided to test all the students using antigen tests with samples of the students' saliva.

Since all teachers have forgotten how to use those a long time ago, Adam signed himself up as a volunteer to help with the tests.

He received saliva samples from N students (for privacy reasons, he is only allowed to know identifiers from 0 to $N-1$) and his task is to determine which samples are positive. Unfortunately, it was too late when he realized that testing all students is an extremely long and boring task. However, he realized that he can do the testing process in a smarter way than by testing the samples one by one. If he mixes a subset of the samples and tests this mix, he will find out whether all of the samples in this mix are negative or if at least one of them is positive. He could use this to reduce the number of tests he needs to do!

Since there is enough saliva in each sample, he can test a sample as many times as he wants. Furthermore, the tests are absolutely precise, so it never happens that different tests yield different results for the same sample.

Under those conditions, he would like to optimize the process to use as few tests as possible. However, he's busy with the testing, so the optimization of the process is up to you.

From local statistics, Adam was able to figure out that the probability that any given sample is positive is equal to P, and that one sample being positive or negative isn't influenced by the positivity or negativity of any other samples. Perhaps you can use this to optimize what tests he performs?

Communication

This task is interactive.

Your program will be executed on a number of test cases. As part of one test case, and therefore during one execution of your program, you will have to solve T different scenarios. The value of N and P is the same across all scenarios, but which students' samples are positive will (most probably) be different in each scenario.

You can either implement the required protocol yourself, or use the provided template. You can find the template in CMS as a task attachment called `template.cpp`.

The protocol

First, your program should read a line from the standard input containing an integer N, a real number P and an integer T separated by spaces — the number of students, the probability of a positive sample and the number of scenarios.

After that, the program can print queries to the standard output. Each query should be a single line containing `Q`, a space, and a string s of length N, where s_i is `1` if Adam should add the i-th student's sample to the test, and `0` otherwise. After printing this line, the program should flush the standard output and then read one character on a single line which will be `P` if at least one sample from the tested group is positive, and `N` otherwise.

The program can also print the answer as a single line to the standard output containing `A`, a space, and a string s of length N, where s_i is `1` if i-th student's sample is positive, and `0` otherwise. After printing this line, the program should flush the standard output and then read one character on a single line.

If the line contains `C`, then your answer was correct. In that case the program may start performing queries about the next scenario, or, if this was your T-th answer, exit.

If the line contains `W`, then your answer wasn't correct. In that case the program should exit immediately.

Please note that exiting after `W` is important for getting the right feedback from CMS. If your program continues running, it may crash or receive some other unsuccessful verdict.

The template

If you are using the implementation of the protocol in `template.cpp`, you need to implement the function `std::vector<bool> find_positive()`. This function will be called once for each scenario. It must return a vector of Booleans of length N, where the i-th element is `true` if and only if the i-th student's sample is positive.

To do so, you make use of the function `bool test_students(std::vector<bool> mask)`. This function performs a test on a subset of samples. Its only argument is a vector of Booleans of length N, where the i-th element is true if the i-th sample should be added to the mix. It returns `true` if (and only if) at least one of the samples in the mix is positive.

You can also use the global variables `N` and `P` containing N and P from the statement. You may do any initialization in the `main` function after the first call to `scanf`.

Inputs

The judge for the task is not adaptive, which means that the positivity of individual samples is determined prior to running your program. Furthermore, whether any given sample is positive has been determined independently with a probability of P using a fair random number generator.

Subtasks and grading

There are two subtasks.

First subtask (10 points)

- $N = 1\,000$
- $T = 1$
- $0 \leq P \leq 1$

A solution is accepted if it answers correctly and uses at most $2 \cdot N$ queries on every test case.

Second subtask (90 points)

- $N = 1\,000$
- $T = 300$
- $0.001 \leq P \leq 0.2$

This subtask uses partial scoring.

If your answer on any scenario is wrong, you will receive zero points. Otherwise, the number of points for a given test case will be determined based on the average number of queries per scenario. In general, a smaller number of queries yields a larger number of points. Let Q denote the average number of queries used by your program across all scenarios, rounded down to one decimal place. For each test case, we have computed a value F (see below). The score on a given test case will be computed according to the following rules:

- If $Q > 10 \cdot F$ you will receive 0 points (wrong answer).
- If $F < Q \leq 10 \cdot F$, the number of points will be determined by the following formula:

$$90 \cdot \frac{F}{F + 4 \cdot (Q - F)}$$

- If $Q \leq F$, you will receive the full 90 points.

Your solution will be graded on several test cases with different values of P. The total number of points you'll receive will be the minimum number of points across all test cases (i.e., across all probabilities P).

There are the following test cases:

P	F
0.001	15.1
0.005256	51.1
0.011546	94.9
0.028545	191.5
0.039856	246.3
0.068648	366.2
0.104571	490.3
0.158765	639.1
0.2	731.4

The judging system will provide feedback for every test case. This feedback will include the value of Q of your solution on each test case where you obtain a non-zero score.

Sample interaction

Here is a sample interaction with the grader. Please note that these values of N and T can't appear in any subtask. Do not forget to flush the output after every line.

Your input	**Your output**
`10 0.4 2`	
	`Q 1000000000`
`P`	
	`Q 0000001000`
`P`	
	`Q 0000000001`
`P`	
	`Q 0111110110`
`N`	
	`A 1000001001`
`C`	
	`A 0000000000`
`W`	

The program solved the first scenario correctly, but not the second one, as the correct solution was `1100010010` (which the program couldn't have known, as it didn't perform any queries). Even if there were another scenario, the program should immediately terminate.