Flash memory

1 sec / 10 sec **1 GB**

A microcontroller chip has B bits of built-in flash memory. We need to store and update an M-bit variable in there. Flash memory has a limitation that bits can be changed from `0` to `1` individually, but changing from `1` to `0` is possible only by erasing the whole memory. The memory can be erased only a limited number of times before the chip is worn out and has to be replaced. Therefore, it is desirable to be able to write as many values as possible before erasing.

Your task is to devise an efficient way of storing values to flash memory so that the current value can always be retrieved. More precisely, your program has to implement two operations:

- Writing a value: The inputs for this operation are the current state of the memory and the new value to be written and you have to output the new state of the memory.
- Reading a value: The input for this operation is the state of the memory after some number of write operations and you have to output the value that was written in the last write operation.

Your reading and writing operations can not exchange information between them in any other way besides reading the current state of the memory from input and the writing operation changing some bits (possibly none) from `0` to `1` before writing the new state to output.

Interaction. This is an interactive task. When your program starts, the first line of input contains integer T, where $T = 0$ means your program will write values to memory and $T = 1$ means it will read values from memory. The second line contains the integers B and M. The following lines describe the operations.

For both writing and reading, the first line of each operation contains integer C, where $C = 0$ means there will be no more requests and your program should exit, but $C = 1$ means your program should continue.

- For $T = 0$ and $C = 1$, the second line contains two space-separated strings: the current state of the memory as a sequence of B bits and the new value to be written as a sequence of M bits. If your program can write the new value to the memory by only changing some bits from `0` to `1`, it should first output the integer 1 and on the next line the new state of the memory as a sequence of B bits. If your program can't write the new value to the memory, it should output the integer 0.
- For $T = 1$ and $C = 1$, the second line contains a single string: the state of the memory as a sequence of B bits and your program should output a single string: a sequence of M bits, the value that was last written to the memory.

Example. Input Output

```
0
6 2
1
111111 00
               0
1
000000 11
               1
               110000
0
```

In this example, your program is started to write 2-bit values into 6-bit memory. The first request is to write the value `00`, which your program is not able to do. The second request is to write the value `11`, which your program is able to do. Note that the current state of the memory for the second request does not match your program's output after the first request.

Example. Input Output

```
1
6 2
1
110000
                  11
1
110100
                  01
0
```

In this example, your program is started to read 2-bit values from 6-bit memory. The first request is to read from the memory state `110000`, from which your program extracts the value `11`. The second request is to read from the memory state `110100`, from which your program extracts the value `01`.

Remark. To ensure your responses are delivered to the grading environment, you have to flush the output buffer after each response (also note that the output always ends with a newline):

Language	Command
C	`fprintf(stdout, "1\n%s\n", s);` `fflush(stdout);`
C++	`cout << 1 << "\n" << s << endl;`
Java	`System.out.println("1");` `System.out.println(s);` `System.out.flush();`
Python	`sys.stdout.write("1\n{0}\n".format(s))` `sys.stdout.flush()`

Testing. For each test case, 4 instances of your program will be started simultaneously, 2 for writing and 2 for reading. The memory and CPU time limits are for all these instances combined. **Any deliberate attempt to pass data out-of-band between these instances is considered cheating and will be cause for disqualification.**

Several blocks of memory, each B bits, will be initialized with zeros. Then writing and reading operations will be performed in some feasible order.

During a writing operation, a writer instance is given the current state of one of the blocks and a value to write into it. You may assume the values to be written have been chosen uniformly randomly from the range $0 \ldots 2^M - 1$ and independently of everything else. If your program is able to write the value, the state of the block will be set to the one returned by your program. If your program can't write the value, that block won't be part of any further writing operations.

During a reading operation, a reader instance is given the state of a block after a successful writing operation. It is checked whether the value returned by your program is the same as what was supposed to be written during the write operation. The read operation will be performed exactly once on the output of each successful write operation.

Grading. In each test group, your program's score is proportional to the average number of values written in tests within this group. More precisely, if your program can write V values per block on average, its score will be $100 \cdot V/P\%$ of the value of the test group, where P is as given below. When your program returns a wrong value in any reading operation, the score for the whole group will be zero. For any other failure, the number of values read in that test will be counted as zero.

The test groups satisfy the following conditions:

1. (5 points) $B = 16$, $M = 8$, $P = 4.06244502449506962405 6$.
2. (5 points) $B = 32$, $M = 8$, $P = 12.26490484130096483417 7$.
3. (5 points) $B = 32$, $M = 16$, $P = 4.12959151370778480200 6$.
4. (5 points) $B = 64$, $M = 8$, $P = 30.03927789426882890003 0$.
5. (5 points) $B = 64$, $M = 16$, $P = 12.95314809421736043271 5$.
6. (5 points) $B = 64$, $M = 32$, $P = 4.07355978823366150153 7$.
7. (5 points) $B = 128$, $M = 8$, $P = 69.77789222892874754877 5$.
8. (5 points) $B = 128$, $M = 16$, $P = 34.73179127514363524097 6$.
9. (5 points) $B = 128$, $M = 32$, $P = 13.95078898770563890866 3$.
10. (5 points) $B = 128$, $M = 64$, $P = 4.03991821060480013390 7$.
11. (5 points) $B = 256$, $M = 8$, $P = 174.46804708607103851145 3$.
12. (5 points) $B = 256$, $M = 16$, $P = 82.22261415140417733455 4$.
13. (5 points) $B = 256$, $M = 32$, $P = 37.62938226976920648891 6$.
14. (5 points) $B = 256$, $M = 64$, $P = 14.26346228205414057768 6$.
15. (5 points) $B = 256$, $M = 128$, $P = 4.01556909389394343085 9$.
16. (5 points) $B = 512$, $M = 16$, $P = 204.74624212741034617022 1$.
17. (5 points) $B = 512$, $M = 32$, $P = 91.77859514807311153984 7$.
18. (5 points) $B = 512$, $M = 64$, $P = 39.23027924214593871262 1$.
19. (5 points) $B = 512$, $M = 128$, $P = 15.00000000216767226860 1$.
20. (5 points) $B = 512$, $M = 256$, $P = 4.00542327711105546887 6$.

Additionally, all test cases satisfy $N \cdot B \leq 120\,000$, where N is the maximal number of write operations your program may be asked to perform.

During the contest, your solution will be graded on a small set of tests for each group. After the contest, your last submission and the submission that scored best on the small test set will be tested on a bigger test set and the best of those two will be your final score for this task. **The score you see in CMS during contest is not your final score.** The purpose of this is to increase the accuracy of your score. Grading on full test set during contest would be too slow.