Kattis

AUPC Practice Contest 1

2025-03-21

Problems

A. 24 Game
B. Peningar
C. Credit Card Payment
D. Morse Code Palindromes
E. 2048
F. Access Denied

Advice, hints, and general information

- Your solution programs should read input from standard input (e.g. System.in in Java or std::cin in C++) and produce output on standard output (e.g. System.out in Java or std::cout in C++). Anything written on standard error will be ignored. For further details and examples, please refer to the documentation in the help pages for your favorite language on Kattis.
- If you think some problem is ambiguous or underspecified, you may ask the judges for a clarification request through the Kattis system. The most likely response is “No comment, read problem statement”, indicating that the answer can be deduced by carefully reading the problem statement or by checking the sample test cases given in the problem.

24 Game

CPU TIME LIMIT	MEMORY LIMIT
1 second	1024 MB

In his unending goal to become the greatest programmer of all time, Gustav has decided to go to Singapore. Curiosity leads him into the game room, where some of his new friends are playing cards. "What game are you playing?" he asks. His friends go on to tell him about a seemingly ancient mathematical card game called twenty four. The goal is to create the number T, typically 24, using only the numbers on the C cards placed on the table, the four basic arithmetic operators (+,-,*,/) and parentheses. Gustav can also change the order of the cards. Note that unary operators are not allowed (i.e., $1 \cdot -2$ is invalid). Cards with numbers on them take on that value. Ace represents one, Jack represents 11, Queen represents 12 and King represents 13.

Since Gustav's new friends are incredibly experienced and beat him every single round, help him write a program that can play for him.

Input

The first line contains two positive integers C and T ($1 \leq C \leq 6$, $1 \leq 10^3$), the number of cards and the target number respectively. The second line contains C integers $c_1, c_2, \ldots c_C$ ($1 \leq c_i \leq 13$), the cards placed on the table.

Output

Print any solution that equals T. There is guaranteed to exist at least one solution.

Points

Your solution will be tested on several test case groups. To get the points for a group, it must pass all the test cases in the group.

Group	Points	Constraints
1	20	$C = 4, T = 24$
2	20	$C \leq 4$
3	60	No additional constraints

Sample Input 1

```
4 24
1 12 1 12
```

Sample Output 1

```
(((12-1)+12)+1)
```

Sample Input 2

```
4 24
1 2 3 4
```

Sample Output 2

```
(((4*3)*2)*1)
```

Sample Input 3

```
4 24
3 3 8 8
```

Sample Output 3

```
(8/(3-(8/3)))
```

Peningar

CPU TIME LIMIT	MEMORY LIMIT
1 second	1024 MB

Image from flickr.com

Tómas has found himself in a strange world. This world consists of n cells arranged in a circle. Thus cells i and $i+1$ are adjacent for $1 \leq i < n$, and cells 1 and N are also adjacent. In each cell there is an a_i amount of money. Tómas starts at cell 1. In each step he moves d cells. When landing in a cell Tómas takes all the money in that cell. Can you find out how much money Tómas will get if he continues walking forward until he can't gather any more money?

Input

The first line of the input contains two integers n, d ($1 \leq n \leq 10^5$, $1 \leq d \leq 10^{14}$), where n is the number of cells and d is how many cells Tómas moves forward with each step.

The next line contains n integers, a_i ($1 \leq a_i \leq 10^9$), denoting how much money is in the i-th cell.

Output

Print a single integer, how much money Tómas will end up with.

Scoring

Group	Points	Constraints
1	25	$1 \leq n, a_i \leq 100, d = 1$
2	25	$1 \leq n, d, a_i \leq 100$
3	50	No further constraints

Sample Input 1

```
4 1
1 1 1 1
```

Sample Output 1

```
4
```

Sample Input 2

```
4 2
1 5 3 5
```

Sample Output 2

```
4
```

Sample Input 3

```
5 3
1 2 3 4 5
```

Sample Output 3

```
15
```

Credit Card Payment

CPU TIME LIMIT	MEMORY LIMIT
1 second	1024 MB

Using credit cards for your purchases is convenient, but they have high interest rates if you do not pay your balance in full each month.

The interest rate is commonly quoted in terms of "annual percentage rate" (APR) which is then applied to the outstanding balance each month. The APR can be converted to a monthly interest rate R. At the end of each month, the monthly interest rate is applied to the outstanding balance and the interest is added to the total balance. Any payment made will be applied to the balance in the following month. The monthly interest is rounded to the nearest cent (rounding up 0.5 cent and above) in the calculations.

You have unfortunately accumulated an outstanding balance B at the end of the month and you can only afford to pay up to some amount M every month. If you do not make any more purchases with the credit card, what is the minimum number of payments needed to completely eliminate the outstanding balance? It is possible that you cannot pay off the balance in 100 years (1 200 payments).

Input

The input consists of multiple test cases. The first line of input is a single integer, not more than 1 000, indicating the number of test cases to follow. Each of the following lines specify the input for one case. Each line contains three positive real numbers separated by single spaces: R, B, and M. The real numbers have two digits after the decimal point, satisfying $R \leq 50.00$ and $B, M \leq 50\,000.00$. R is the monthly interest rate and is specified as a percentage.

Output

For each case, display on a line the minimum number of payments needed to eliminate the outstanding balance. If this cannot be done in at most 1 200 payments, print instead `impossible`.

Sample Input 1

```
11
2.00 100.00 105.00
2.00 100.00 102.00
2.00 100.00 100.00
2.00 100.00 4.00
2.00 100.00 3.00
2.00 100.00 1.00
2.00 100.00 2.00
9.56 5462.50 522.22
12.50 29876.44 33610.99
5.50 1.00 1.05
14.78 40181.09 46119.86
```

Sample Output 1

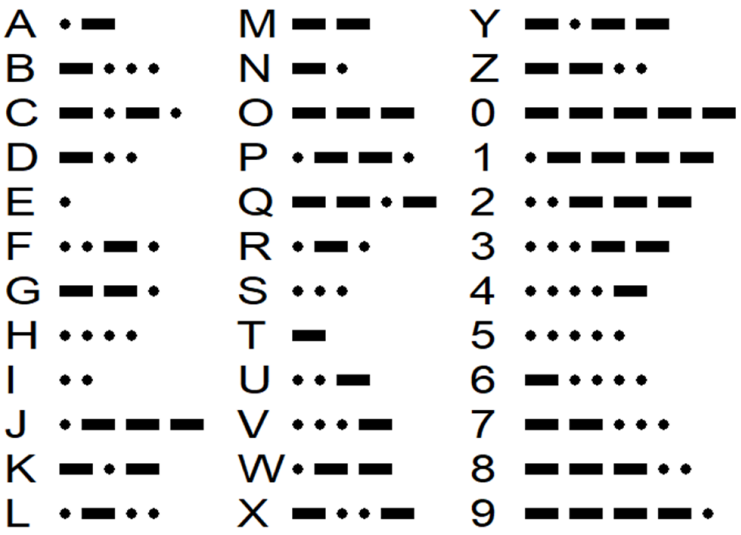
```
1
1
2
36
56
impossible
impossible
impossible
2
2
1
```

Morse Code Palindromes

CPU TIME LIMIT	MEMORY LIMIT
1 second	2048 MB

A *Palindrome* is a word, number, phrase, or other sequence of characters which reads the same backward as forward, such as **madam** or **racecar** or **1881**. Phrase palindromes ignore capitalization, punctuation, and word boundaries. For example: **Madam I'm Adam.**

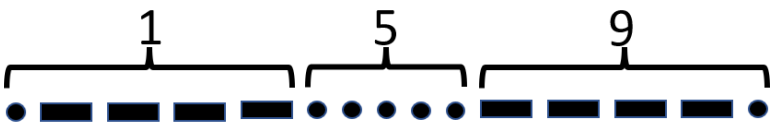
Morse code is a method used in telecommunication to encode text characters as standardized sequences of two different signal durations, called dots and dashes, or dits and dahs. Morse code is named after Samuel Morse, one of the inventors of the telegraph. The international morse code for letters and digits is:



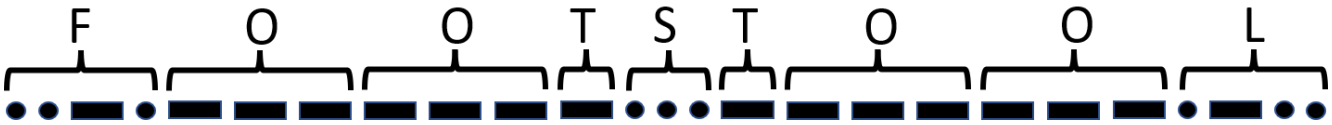
(Note that the code for upper and lower case letters is the same.)

A word, number or phrase is a *Morse Code Palindrome* if the morse code for the letters and digits in the word, number or phrase reads the same backwards and forwards (ignoring spaces between character codes). For example:

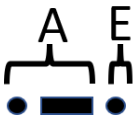
159



Footstool



A E



Determine if an input string is a *Morse Code Palindrome*.

Input

The single input line contains a string of up to 80 ASCII characters, possibly including spaces and other non-alphanumeric printable characters.

Output

The single output line consists of the integer `1` if the input string (ignoring everything but letters and digits) is a *Morse Code Palindrome*. Otherwise the output line consists of the integer `0`. If there are no letters or digits in the input string, the output should be `0`.

Sample Input 1

```
hello
```

Sample Output 1

```
0
```

Sample Input 2

```
159
```

Sample Output 2

```
1
```

Sample Input 3

```
Madam I'm Adam
```

Sample Output 3

```
0
```

Sample Input 4

```
footstool
```

Sample Output 4

```
1
```

Sample Input 5

```
SOS
```

Sample Output 5

```
1
```

Sample Input 6

```
A E
```

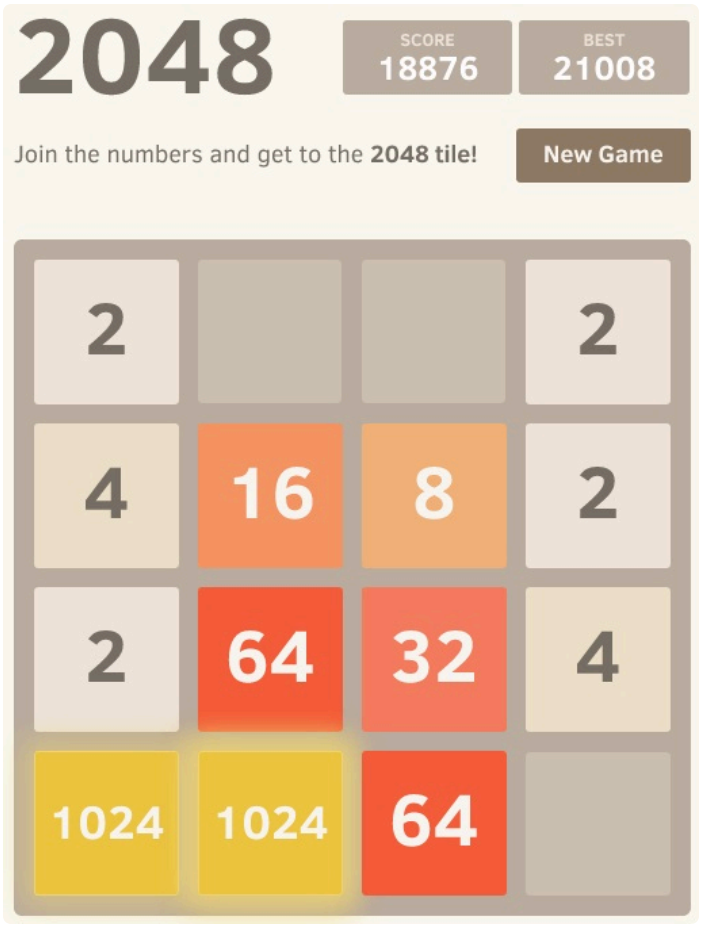
Sample Output 6

```
1
```

2048

CPU TIME LIMIT	MEMORY LIMIT
1 second	1024 MB

2048 is a single-player puzzle game created by Gabriele Cirulli[1]. It is played on a 4×4 grid that contains integers ≥ 2 that are powers of 2. The player can use a keyboard arrow key (left/up/right/down) to move all the tiles simultaneously. Tiles slide as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid. If two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles that collided. The resulting tile cannot merge with another tile again in the same move. Please observe this merging behavior carefully in all Sample Inputs and Outputs.



screenshot taken from http://gabrielecirulli.github.io/2048/

Input

The input is always a valid game state of a 2048 puzzle. The first four lines of input, that each contains four integers, describe the 16 integers in the 4×4 grid of 2048 puzzle. The j-th integer in the i-th line denotes the content of the cell located at the i-th row and the j-th cell. For this problem, all integers in the input will be either {0, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024}. Integer 0 means an empty cell.

The fifth line of input contains an integer 0, 1, 2, or 3 that denotes a left, up, right, or down move executed by the player, respectively.

Output

Output four lines with four integers each. Two integers in a line must be separated by a single space. This describes the new state of the 4×4 grid of 2048 puzzle. Again, integer 0 means an empty cell. Note that in this problem, you can ignore the part from the 2048 puzzle where it introduces a new random tile with a value of either 2 or 4 in an empty spot of the board at the start of a new turn.

Sample Input 1

```
2 0 0 2
4 16 8 2
2 64 32 4
1024 1024 64 0
0
```

Sample Output 1

```
4 0 0 0
4 16 8 2
2 64 32 4
2048 64 0 0
```

Sample Input 2

```
2 0 0 2
4 16 8 2
2 64 32 4
1024 1024 64 0
1
```

Sample Output 2

```
2 16 8 4
4 64 32 4
2 1024 64 0
1024 0 0 0
```

Sample Input 3

```
2 0 0 2
4 16 8 2
2 64 32 4
1024 1024 64 0
2
```

Sample Output 3

```
0 0 0 4
4 16 8 2
2 64 32 4
0 0 2048 64
```

Sample Input 4

```
2 0 0 2
4 16 8 2
2 64 32 4
1024 1024 64 0
3
```

Sample Output 4

```
2 0 0 0
4 16 8 0
2 64 32 4
1024 1024 64 4
```

Sample Input 5

```
2 2 4 8
4 0 4 4
16 16 16 16
32 16 16 32
0
```

Sample Output 5

```
4 4 8 0
8 4 0 0
32 32 0 0
32 32 32 0
```

Sample Input 6

```
2 2 4 8
4 0 4 4
16 16 16 16
32 16 16 32
2
```

Sample Output 6

```
0 4 4 8
0 0 4 8
0 0 32 32
0 32 32 32
```

Footnotes

1. Based on 1024 by Veewo Studio and conceptually similar to Threes by Asher Vollmer.

Access Denied

CPU TIME LIMIT	MEMORY LIMIT
2 seconds	1024 MB

IBM 7090 console (Public Domain)

Computer passwords have been around for a long time. In fact, 60 years ago CTSS was one of the first computers with a password. The implementation of this was very simple. In CTSS the password was stored in plain text in a file on disk. When logging in, the user would enter a password. The computer would then compare the password to the password on disk. If the comparison failed, it would deny access, if it succeeded, access would be allowed. Researchers at MIT were quick to discover several security flaws in this password system. We will explore one of them, the timing attack.
In a timing attack, we exploit that we can deduce a computation path from the time it takes to do the computation. In CTSS the password check was done using a simple string matching algorithm, similar to this:

```
bool CheckPassword(string pwd1, string pwd2) {
    if (pwd1.Length != pwd2.Length) {
        return false;
    }
    for (int i = 0; i < pwd1.Length; i++) {
        if (pwd1[i] != pwd2[i]) {
            return false;
        }
    }
    return true;
}
```

For the purpose of this problem, we will use a (very) simplified timing model and the above algorithm. The timing model looks as follows:

- A branching statement (if or for) takes 1 ms.
- An assignment, or update of a memory address takes 1 ms.
- A comparison between two memory addresses takes 3 ms.
- A return statement takes 1 ms.

The password consists of between 1 and 20 English letters, upper or lower case, and digits.

Interaction

This is an interactive problem. Your submission will be run against an *interactor*, which reads the standard output of your submission and writes to the standard input of your submission. This interaction needs to follow a specific protocol:

- Your program first sends a password string, consisting of between 1 and 20 English letters, upper or lower case, and digits.
- Depending on if the password is correct, the interactor then responds with either:
  - If the password is correct; "`ACCESS GRANTED`". Your program should then exit cleanly.
  - If the password is incorrect; "`ACCESS DENIED (`t` ms)`", where t is the time it took to verify the password in ms. Your program can then make another guess.

Make sure you flush the buffer after each write. You can guess at most 2 500 times. A testing tool is provided to help you develop your solution.

Read	Sample Interaction 1	Write

```
A
```

```
ACCESS DENIED (5 ms)
```

```
HunFhun
```

```
ACCESS DENIED (41 ms)
```

```
Hunter1
```

```
ACCESS DENIED (68 ms)
```

```
Hunter2
```

```
ACCESS GRANTED
```