Arithmetic Automaton

Easy

An automaton is a self-operating machine. The automaton in question consists of an array of three or more cells which are joined cyclically (the first cell and last cell have each other as neighbours). Each cell holds a decimal number between 0 and 9. The value of each cell is updated in synchronization with a periodic tick. The new value of each cell is a function of the old value in that cell as well as the value in that cell's neighbours.

Your task is to simulate the operation of the automaton and determine its state after a certain number of ticks. The state of each cell can be determined according to the following functions:

L = value of left neighbour, S = value of self, R = value of right neighbour, | | = absolute magnitude, % = modulus operator, == = equality operator, != = inequality operator

$S < L < R$; $S = |S * (L + R)| \% 10$

$S < R < L$; $S = |S * (L – R)| \% 10$

$L < S < R$; $S = |S + (L + R) * (L – R)| \% 10$

$L < R < S$; $S = |S - (L + R) * (L – R)| \% 10$

$R < S < L$; $S = |S + 2L*R| \% 10$

$R < L < S$; $S = |S - 2L*R| \% 10$

R == L & R!=S; S = S

S == R & R!=L; S = L

S == L & R!=L; S = R

S == R & R== L; S = |S + R + L| % 10

Input data consists of two lines. The first line describes the state of each cell in the automaton (and by extension, the number of cells in automaton, which will be no more than 64) while the second line describes the number of ticks that must be simulated. Output should be in the form of a single line describing the state of the automaton after the prescribed number of ticks just as it was read in.

Example Input:

```
459623

7
```

Example Output:

```
081803
```

Bender’s Blunder

Easy

A team of engineers is designing a new type of robot for their capstone project. Unfortunately, they ran out of time and weren’t able to finish implementing self-awareness before they had to demonstrate it. However, they were able to control it remotely. In order to demonstrate the robot the team had it navigate a maze by remote control.

The maze consists of a square grid of cells. The robot can navigate from one cell to another by moving forward, moving backward, turning left, or turning right.

The maze includes a number of obstacles including walls, pits, and traps. There is also an entry point and an exit point.

Input consists of three parts.

The first is a whole number n which is the dimension of the field.

The second is an n*n grid with each grid cell represented by a character. Each line of input data corresponds to a single row on the grid and each character on the line represents a single cell on that row. The character legend is as follows:

E = Entry point. Note that this will always be placed along one edge of the maze; the robot’s initial orientation will be as if it had just walked into the maze from outside.
[X] = Exit point. If the robot reaches the exit point, the demonstration is over.
[P] = Pit. If the robot falls into a pit, the demonstration is over.
[W] = Wall. If the robot runs into a wall, it cannot proceed in that direction. Note that the robot can attempt to proceed into a wall repeatedly if the instructions call for it.
[T] = Trap. Traps are dangerous to humans, but are rather useless against a robot constructed from a zinc-titanium alloy.
[ ] = Traversable ground (this is a white space character)
Second is a sequence of instructions that must be followed. Instructions are concatenated and are in a character stream format.
[Fx] = Move forward x tiles
[Bx] = Move backward x tiles
[Lx] = Turn left x times
[Rx] = Turn right x times
Output is a sequence of exclamations and curses, each on one line

If the robot reaches the exit point, print the phrase “Aww Yeah!”

If the robot runs into a wall, print the phrase “Ow!” each time

If the robot runs into a trap, print the phrase “Bite my shiny metal ass!”

If the robot falls into a pit, print the phrase “Well, I’m boned”

Example Input:

10

```
WWWWWWXWWW
W    W W W
W      WWW
W        W
WTTT     W
W        W
W  PWWWWWW
W T      W
WW       W
WWWEWWWWWW
F2L1F1R1F3R1F3L1F5R1F1L1F2
Example Output:
Bite my shiny metal ass!
Bite my shiny metal ass!
Bite my shiny metal ass!
Ow!
Ow!
Ow!
Aww Yeah!
```

Canadian Caper

Easy

Captain Canuck has enlisted the help of a team of McMaster Engineering students to help rescue some Waterloo students that have been kidnapped by Doctor Evil. Perplexed as they were about how a bunch of elite undergraduates attending such a prestigious university managed to get captured by a homeless mathemagician (much less why) they none the less acquiesced to Captain Canuck's request.

After locating Doctor Evil behind a nearby dumpster, the evil doctor agreed to reveal the location of the Waterloo students as long as the McMaster students were able to decipher a numeric code using some supplied instructions in a ridiculously short period of time. Once provided with the numbers, they would not have sufficient time to decipher the code by hand, but the doctor would allow them to write a computer program to do it.

The doctor would provide the students with nine sequences of nine letters and whole numbers. Each sequence was in order, but the sequences themselves were out of order.

Each sequence was to be read into a 3x3 matrix. Matricies that contained letters were dependent on a previous sequence, so a matrix with only numbers would have to be solved first. Letters are symbolic for the determinant of a previously solved matrix. For example, any matrix that includes the letter A has the determinant of the first matrix substituted in place and any matrix that includes the letter B has the determinant of the second matrix substituted in place.

The passcode demanded by the doctor is the determinants of the matricies in the order of dependence.

NOTE: Not all matricies will be dependent on all previously matricies, but there will never be two matricies with the same dependencies.

Splitting Straws

Easy

An incredibly bored engineer has a number of drinking straws that he or she doesn't need and decides to have some fun with. The engineer measures the straws such that they are all a whole multiple of some unit length.

The engineer then performs the following steps:

1. The engineer takes all straws of even length, and cuts them in half if and only if both of the resulting halves would be of even length as well.
2. The engineer then takes all straws of odd length, and cuts from each a length equal to the shortest odd-length straw.
3. The engineer then repeats steps #1 and #2 until the remaining portion of each straw cut in step #2 is shorter than the amount that was cut from them, no odd-length straws are remaining, or all odd straws are of the same length

NOTE: Termination conditions depend on the outcome of step #2. Each step must be run at least once, even if it has no effect.

Your task is to determine how many straw segments the engineer is left with at the end.

Input is in the form of two lines. The first line contains a single number, this is the number of straws that the engineer starts with. The second line contains a sequence of whitespace delimited numbers representing the length of each straw.

Output should be in the form of a single whole number representing the total number of straw segments.

Example Input:

7

15 20 8 9 4 10 3

Example Output:

15

Vindictive Vending

Easy

The vending machines on campus are notorious for short changing hard working engineering students. While one should never attribute to malice that which can be adequately attributed to stupidity, any sufficiently shocking display of stupidity is indistinguishable from malice. Ergo, one can only assume that the faulty vending machines are a ploy on part of the university administration to extract supplementary tuition from students.

Each vending machine is capable of proving nickels, dimes, quarters, loonies, and toonies as change. It contains a finite number of each. In addition to this, it accepts $5 and $10 bills as methods of payment.

Your task is to calculate the minimum number of coins to return to students when providing change for a purchase. You must also reject bills if adequate change cannot be immediately provided.

Input is in the form of several sets of numbers, one set per line. The first set of numbers denotes the initial supply of nickels, dimes, quarters, loonies, and toonies in that order. Every subsequent line describes a purchase. The first number is the purchase amount, and the second number is the provided payment.

There should be one output line per purchase. If the purchase succeeds, the line should include the number of nickels, dimes, quarters, loonies, and toonies returned. If the purchase cannot be completed, print “Insufficient Change”.

Example Input:

Scrambled Sequence

Medium

A ~~soulless demon~~ mathemagician wrote down on a piece of paper his favourite sequence of six unique numbers. On a separate piece of paper he then wrote down six copies of that same sequence. In each copy of the sequence, exactly one digit in the sequence was moved to a different position in the sequence. Each digit is never moved more than once. The mathemagician then challenged a team of students to reconstruct the original sequence from the six altered sequences.

Input is in the form of five lines each containing five white space delimited numbers. Output should be in the form of a single line containing six white space delimited numbers.

Example Input:

```
1 2 6 3 4 5

1 2 3 5 4 6

1 5 2 3 4 6

3 1 2 4 5 6

2 3 4 5 6 1

1 3 4 2 5 6
```

Example Output:

```
1 2 3 4 5 6
```

Tricky Triangle

Hard

A 3-dimensional Cartesian space contains a point light source, a triangle, and a plane.

The point light source is described by a single point in space, the triangle by three points in space, and the plane by three points with one point on each coordinate axis.

The triangle is located between the point and the plane.

The point source casts a light on the triangle and the plane; the triangle obstructs the light and casts a shadow on the plane.

Your task is to calculate the surface area of the shadow cast by the triangle onto the plane.

Input is in the form of a series of points, one point per line. Each point is a triple of real numbers describing each point's X, Y, and Z coordinates respectively. The first point describes the point light source; the next three points describe the triangle; the final three points describe the plane.

Output is in the form of a single real number, rounded to three decimal places.

NOTE: To avoid precision and rounding errors, use double precision floating point data types throughout your program

Example Input:

50.0 50.0 50.0

25.0 25.0 20.0

30.0 30.0 15.0

20.0 25.0 10.0

5.0 0.0 0.0

0.0 5.0 0.0

0.0 0.0 5.0

Example Output:

43.923

Sticks and Stones

Medium

Having recently recalled a popular childhood rhyme about sticks and stones being able to break bones, Doctor Evil decides to construct a weapon out of just those to do just that. Unfortunately, as a result of budget cuts to the Department of Evil Genius Engineering (EGE) he will have to do so with limited resources. Doctor Evil has at his disposal a large quantity of massless rods of unit length and a handful of small spheres of unit diameter but varying density.

Doctor Evil places a single sphere at the origin and begins to build outward from there by following a procedure.

Each sphere has a number of mounting points to which additional spheres may be connected via a massless rod. The number of mounting points is equal to *n + 2* where *n* is the number of rods between that sphere and the origin sphere; for the origin sphere, n is equal to zero. Each mounting point is located 2π/n radians apart, and the mounting point at θ = 0 is always used to connect to the parent sphere (with the exception of the sphere at the origin which has no parent). Note that the angle θ = 0 is relative to each sphere, not absolute in space.

Child spheres are attached to the mount points on parent spheres in order of increasing relative angle (scanned counter-clockwise) starting at θ = 0 and ending at θ = 2π. This logic is applied recursively (the child at point 4π/3 on the sphere attached to the origin at θ = 0 is to be populated before the child at point 2π/3 on the sphere attached to the origin at θ = π), and all mounting points on children of a given distance *n* from the origin sphere must be populated before any mounting points on children of a distance *n+1* from the origin sphere are populated.

If there are insufficient spheres to fully populate all mounting points, then the remainder are to remain unpopulated.

Your task is to use the mass and absolute position of each sphere to calculate the center of mass of the entire device.

Input consists of two lines. The first line contains only a single whole number which is the number of spheres in the device. The second line contains a sequence of whitespace delimited real numbers indicating the mass of each sphere. Spheres are attached to the device using the logic described above in the order that they appear in the input.

Output should be in the form of two white space delimited real numbers describing the X and Y position of the center of mass. Use double precision floating point arithmetic throughout and round to three decimal places.

NOTE: To avoid precision and rounding errors, use double precision floating point data types throughout your program

Example Input:

```
12
1 23 11 45 12 5 8 19 40 55 23 34
```

Example Output:

```
1.531 -0.287
```

McMaster Microprocessor

Medium

After many years of using the outdated PIC24HJ32GP202, the department of Electrical and Computer Engineering has commissions a custom built little-endian microprocessor called the MACXLE9000-1280. The MACXLE9000 has eight general purpose registers (0-7), each of which is 32-bits in size. There is also an onboard 1KiB ROM which provides 1024 byte-addressable storage locations. The microprocessor supports the following instructions:

# = literal

% = register

@ = memory location

Move Instructions:

s operand may be literals, registers, or memory locations. t operand may be a register or memory location. One of s or t must be a register.

MOVE	t,s	-	Move a 32-bit word from source s to target t

Example: MOVE %4,#256 Move the literal number 256 (decimal) into register number 4

NOTE: The microprocessor is little-endian, which means that the least significant byte in a word occupies the least significant address in memory. Your microprocessor must properly handle unaligned moves

Logic Instructions:

s operands may be literals or registers. t operand must be a register

AND	t,s0,s1	-	Perform logical AND on s0 and s1, store the result in t
OR	t,s0,s1	-	Perform logical OR on s0 and s1, store the result in t
XOR	t,s0,s1	-	Perform logical XOR on s0 and s1, store the result in t
NOT	t,s	-	Perform logical negation on s, store the result in t

Example: NOT %1,%0

Arithmetic Instructions:

s operands may be literals or registers. t operand must be a register

ADD	t,s0,s1	-	Add s1 to s0, store the result in t
SUB	t,s0,s1	-	Subtract s1 from s0, store the result in t
MUL	t,s0,s1	-	Multiply s0 and s1, store the result in t
DIV	t,s0,s1	-	Divide s0 by s1, store the quotient in t
REM	t,s0,s1	-	Divide s0 by s1, store the remainder in t

Example: ADD %3,%2,#90

NOTE: For simplicity, there is a whitespace between the instruction and the first operand, and a comma between each operand

NOTE2: Literals are expanded to 32-bits and padded with zeroes if need be

Your task is to print the contents of the eight general purpose registers (as decimals) after running the program.

Input is in the form of a sequence of instructions, one instruction per line.

Output is in the format of eight white space delimited integers representing the eight general purpose registers (0 through 7 respectively) as **unsigned integers**. Omit leading zeroes.

Example Input:

```
MOVE %0,#0
MOVE %1,#1
MOVE %2,#2
MOVE %3,#3
MOVE @0,%0
MOVE @1,%1
MOVE @2,%2
MOVE @3,%3
MOVE %0,@0
MOVE %1,#90
MOVE %2,#5000
MUL %3,%1,%2
OR %4,%0,%3
ADD %5,%4,#1
MOVE @1020,%5
MOVE %6,@1020
DIV %7,%6,#5
REM %6,%4,#3
```

Example Output:

```
50462976
90
5000
450000
50781648
50781649
0
10156329
```

McMaster Machine

Hard

The McMaster Machine is a communication device that enables users to communicate securely using synchronous encryption. It’s not very good, but it works.

Transmitting a message using the McMaster Machine occurs in three steps: key expansion, message encoding, and message encryption.

Step 1: Key Expansion

Key expansion starts with the key, which is a 64-bit random number in the form of eight individual bytes. An 8x8 matrix is created and the eight bytes are placed into the first row of the matrix in order. In each subsequent row, the eight bytes are rotated cyclically to the left by one byte.

Then, in each cell, the bits are rotated cyclically to the left by the row number.

Next, the two most significant bits in each cell are set to 0.

Finally, the value of each cell is XORed according to its row and column position. There are eight rows (0-7) and eight columns(0-7) each of which are encoded onto three bits. The row position and column position are concatenated to form a 6-bit value unique to each cell which is then XORed against the value in that cell (aligned to the least significant bit).

Step 2: Message Encoding

In order to allow transmitted messages to include basic formatting including non-printable characters, the McMaster Machine encodes text into base-64. Base-64 encodes three bytes (values 0-255) into four text characters. The three bytes are concatenated to form a 24-bit word which is then broken up into four 6-bit words (with two leading zeroes). Each of the 6-bit words can then be interpreted as text using the following scheme:

0-25 -> A-Z | 26-51 -> a-z | 52-61 -> 0-9 | 62 -> + | 63 -> /

The message is padded with trailing zeroes so that the total size of the message is a multiple of 48. Since encoding the message turns each set of 3 bytes into 4 characters, it expands a 48 byte message into a 64 byte message that includes only printable characters.

Step 3: Message Encryption

For each block of 64 encoded bytes, each byte in the block is XORed against a value in the key matrix. The index of the value in the key matrix increments column by column and row by row. Block[0] is XORed against key[0,0], block[1] is XORed against key[0,1], and block[8] is XORed against key[1,0] and so on.

Your task is to reverse these steps. Given a 64-bit key and an encrypted message, decrypt the message to find the original.

Input is in the form of two lines. The first line contains the key; it is stored as hexadecimal pairs for easy parsing into individual bytes. The second line contains the encrypted message.

Output should be the decoded message as an ASCII string. It may include newlines!

Example Input:

```
B2 33 7C 8D 9F 2A 1E 63
hkzDD4VQ3mZUM2BF4Qla3OpyvDXLX+mdCFywUw8alvPIRkKv3pe/rWR9JgjF6yrw
```

Example Output:

```
McMaster Programming Challenge Rules
```