2019 FIU High School Programming Competition

Division 2 Problems

H Miles to Kilometers

I Bishop Checking

J Interpreter

K Boggle Me!

L Hamming Distance

M Heir's Dilemma

N Infix to Postfix

O Polynomial Multiplication

P Postfix Evaluation

R Stacking Cups

H – Miles to Kilometers

Problem Statement

Few people know that you can use Fibonacci numbers to approximately convert distances from miles to kilometers. It is well known, however, that one mile equals exactly 1.6 km. So, how does the Fibonacci conversion method work? First, these are the first 13 numbers of the Fibonacci sequence to use as a guide:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Suppose you want to convert 5 miles into kilometers. In the Fibonacci sequence, the number following 5 is 8, so we can say 5 miles = 8 km. Unfortunately, this method doesn't work all the time. For example, to convert 89 miles into kilometers, the next Fibonacci number after 89 is 144, which is not the right answer. The correct answer is 89 × 1.6 = 142.4 km, so the Fibonacci method is off by 1.6.

The Fibonacci method sometimes allows multiple ways to convert miles to kilometers. For example, if we want to convert 6 miles into kilometers, we could partition 6 into the sum 5 + 1, convert both 5 and 1 separately to kilometers, and then add the resulting values. For example, 5 miles converts to 8 km and 1 mile converts to 2 km; their sum equals 8 + 2 = 10. Using this method, we can say 6 miles is approximately equal to 10 km. On the other hand, we could partition 6 miles differently, as 2 + 2 + 2, convert 2 miles to 3 km, and produce the sum 3 + 3 + 3 = 9 km. Which answer (9 or 10) is closer to the exact conversion value in kilometers? The exact conversion formula produces 9.6 km, so 10 turns out to be the best Fibonacci conversion we tried.

It is your task to write a program that finds the best conversion from miles to kilometers using the Fibonacci method we described earlier. The best conversion is the one with the smallest error, and you are to print the absolute value of that error.

Input Description

Each input will consist of a single test case. Your program will be run multiple times on different inputs. The first line if input will be the integer T, indicating the number of miles test values to follow. This is followed by T lines of input ($1 < T < 21$). Each case will be a single line containing one integer number M ($0 < M \leq 1000$).

Output Description

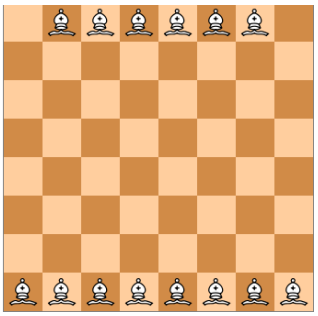
For each test case in the input, your program must print the absolute value of the lowest error for converting the corresponding input to kilometers using the Fibonacci method. Round your answers upward to 2 decimal digits of precision).

Sample Input	Sample Output
3 6 5 12	0.40 0.00 0.20

Source: UVA Online Judge 11780 (mostly rewritten by K. Irvine)

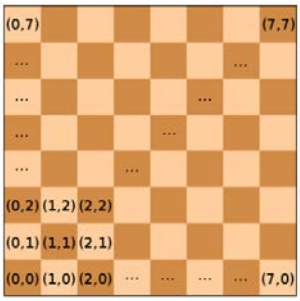
I – Bishop Checking

Problem Statement



The *bishops puzzle* is a classic chess puzzle where you place bishops on an 8 x 8 chess board in such a way that no bishop can attack another bishop. In chess, a bishop can move along any diagonal. One possible solution involving 14 bishops is shown here on the right side of the page. Today, instead of finding solutions to this problem, you need only verify whether a given chess board contains a valid solution.

When referring to specific squares on the board, the bottom-left square is notated (0,0), the x coordinate increases as you move rightward, and the y coordinate increases as you move upward. The figure on the right shows several examples.



Input Description

Each input will consist of a single test case. Your program will be run multiple times on different inputs. Each input begins *N*, an integer that tells you how many bishops will be listed in the input. This is followed by *N* lines, each containing the (x, y) position of a bishop as two integers separated by a single space.

Output Description

The output contains either CORRECT if the given board layout is a solution to the bishops puzzle, and INCORRECT otherwise. Note that the first sample input presented below corresponds to the board shown earlier.

Sample Input 1	Sample Output 1
14 1 7 2 7 3 7 4 7 5 7 6 7 0 0 1 0 2 0 3 0 4 0 5 0 6 0 7 0	CORRECT

Source: Adapted from "queens" problem at open.kattis.com

J – Interpreter

Problem Statement

The Basic programming language, invented by professors at Dartmouth college in the 1970s, provided a great way for beginners to learn how to write computer programs. It used an interpreter, a simple utility program that read each program statement on a single line in a source program and executed it before going to the next statement. Your task is to write a simplified interpreter that reads an input program and displays the value generated by the Print statement. Here, for example, is a program with several variables, assignment statements, and a line that prints the value of *N*:

```
A = 2
B = 1
total = A + B
N = 40 + total
Print N
End
```

Input Description

Each input will consist of a single test case. Your program will be run multiple times on different inputs. Input contains three types of statements: (1) Assignment, (2) Print, and (3) End. Print and End statements will each appear only once. An assignment statement begins with a variable name, followed by an = (equal) sign, followed by an expression. An expression consists of a variable name or constant integer, followed by a binary operator such as +, −, *, or /, followed by another variable name or constant integer. (There will be no negative signs or other unary operators.) In division operations, fractional values are truncated to the next lowest integer. A Print statement consists of the word Print followed by a space, followed by a variable. Variable names are 1 to 10 characters, and are case sensitive. Each input line will be no longer than 80 characters, including spaces.

Output Description

For any Print statement in the program, display the variable name, an equal sign (=), and the value of the variable, all on a single line. There should be one space on either side of the equal sign.

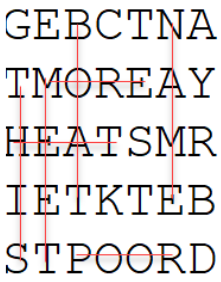
Sample Input	Sample Output
A = 2 B = 1 total = A + B N = 40 + total Print N End	N = 43

K – Boggle Me!

Problem Statement

The Boggle™ game lets you earn points by looking for words in a two-dimensional table of letters. Your task is to do this with a computer, but in a simpler way, because you will only need to verify that a specific set of four-character words was found, using only rows from left-to-right order, and columns in top-to-bottom order.

First, let's start with a list of words we want to find: MORE, HEAT, BOAT, NAME, MEET, POOR, and THIS. Each of these words can be found in this given 5 X 7 table:



It may turn out that some words cannot be found. Your program must determine whether all words in the given list can be found in the table. The table may contain other words not found in the list, but you can ignore them. Use only rows and colums, with no words wrapping around the edge of the grid.

Input Description

Each input will consist of a single test case. Your program will be run multiple times on different inputs. The first line of input will contain N uppercase words ($1 < N < 10$) that you will be looking for in the table. There will be 1 space between each word. The second line of input will contain two integers R ($2 < R < 50$) and C ($2 < C < 100$), which indicate the size of rows and columns in the table that will follow. The following R lines contain the table rows, with one uppercase character per column.

Output Description

If all words can be found in the table, print "YES". Otherwise, print "NO".

Sample Input 1	Sample Output 1
MORE HEAT BOAT NAME MEET POOR THIS 5 7 GEBCTNA TMOREAY HEATSMR IETKTEB STPOORD	YES

Sample Input 2	Sample Output 2
MORE HEAT BOAT NAME MEET POOR THIS 5 7 GEBCTNA TMOREAY HEATSMR AETKTEB STPOORD	NO

L – Hamming Distance

Problem Statement

Wikipedia says: *In information theory, the Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different.* This measure was created by Richard Hamming in 1950.

Your task is to compute the Hamming distance between two binary numbers, given to you as hexadecimal strings. Hexadecimal strings use a hexadecimal digit to represent each 4 bits in a binary string. The following table shows the bits represented by each character:

0 = 0000	4 = 0010	8 = 1000	C = 1100
1 = 0001	5 = 0101	9 = 1001	D = 1101
2 = 0010	6 = 0110	A = 1010	E = 1110
3 = 0011	7 = 0111	B = 1011	F = 1111

Input Description

Input will be two 8-digit hexadecimal strings. The digits A-F will be in upper case.

Output

Print a single integer that indicates the Hamming distance between the binary nmbers represented by the hexadecimal strings.

Sample Input	Sample Output
`BA26F57D` `CD3B689F`	`19`

Source:

M – Heir's Dilemma

Problem Statement

Your favorite uncle, a great lover of rabbits, has passed away, leaving you a large estate. Bank account numbers, locations of safe deposit boxes, and GPS coordinates to buried treasures are all locked in an electronic safe in your uncle's office behind a picture of dogs playing poker. One day he showed you the safe with its 9-digit keypad (digits 1 through 9). He told you he wasn't worried about anyone breaking into his safe because it's equipped with a self-destruct mechanism that will destroy the contents if anyone attempts a forced entry.

The combination is a sequence of six decimal digits. If an incorrect combination is entered the safe enforces a thirty-second delay before accepting another combination. So a brute-force effort to try all six-digit combinations could take months. Your uncle had planned to give you, his sole heir, the combination one day, but due to an unfortunate hang-gliding accident in Kansas, you now must rely on your deductive and programming skills to access the key to your inheritance. Here's what you know:

- The combination *c* is a sequence of six non-zero decimal digits.
- Your mother recalls that she heard your uncle mention that all the digits are different.
- You remember that your uncle once said that the six digit number was divisible by each of its individual digits.

An example satisfying these conditions is 123864: all six digits differ, and you can check that 123864 is divisible by each of 1, 2, 3, 8, 6 and 4. Even with the helpful data, it could take a while to get to open the safe, so the task is likely to be split into several sessions with separate ranges being tested. How many combinations are there to try in the range given?

Input Description

Each input will consist of a single test case. Your program will be run multiple times on different inputs. Input is a single line containing two space-separated integers L and H, where $123456 \leq L < H \leq 987654$.

Output Description

Print one integer, the total number of possible combinations to the safe, where each combination c must satisfy the three constraints above, and lie in the range $L \leq c \leq H$.

Sample Input 1	Sample Output 1
`123864 123865`	`1`

Sample Input 2	Sample Output 2
`198765 198769`	`0`

Sample Input 3	Sample Output 3
`200000 300000`	`31`

Source: 2018 ICPC Mid-Central Regional contest

N – Infix to Postfix

Problem Statement

Computer programs often rely on operator precedence rules in order to compute the values of mathematical expressions. But it's also possible to clarify the operator order by using parentheses. For example, the expression ((a + 2) * (b – 4)) / 3 produces a value that is unambiguous. (We could ask if addition is performed before subtraction in this expression, but the resulting value will be the same.) This is known as a *fully parenthesized infix expression*. The term *infix* means that each binary operator has operands on either side.

But there is another type of expression known as *postfix*, used in some calculators (HP, for example). These types of expressions do not use need to use parentheses to specify the exact order of operations. In a postfix expression, operators follow their operands. For example, [5 2 *] is interpreted as 5 * 2. If there are multiple operators, each operator appears after its last operand. Here are examples, showing how postfix compares to parenthesized expressions:

Postfix	Infix
`5 2 *`	`5 * 2`
`6 3 + 5 * 2 –`	`((6 + 3) * 5) – 2)`
`4 3 2 * +`	`4 + (3 * 2)`

Input Description

Each input will consist of a single test case. Your program will be run multiple times on different inputs. Input contains a fully parenthesized infix expression containing single integer constants, variables, parentheses, and operators (+, −, *, /). A single space will always separate an operator from its neighboring operands. Variable names will always be single alphabetic letters.

Output Description

Output the postfix expression that is equivalent to the given infix expression, assuming that the infix expression is read from the left side to the right side. Insert a single space between all operators and operands.

Sample Input 1	Sample Output 1
`(A + B) * (C – D)`	`A B + C D - *`

Sample Input 2	Sample Output 2
`((4 * y) + (6 – x)) * 2`	`4 y * 6 x - + 2 *`

O – Polynomial Multiplication

Problem Statement

In this program, you're asked to multiply two polynomials, each represented by list of integer coefficents. For example, the list [1 0 5] indicates the polynomial $1 + 0x + 5x^2$, and the list [0 −2] indicates the polynomial $0 - 2x$. Multiplying these two polynomials produces $-10x^3 + 0x^2 - 2x + 0$, if we show all terms.

Input Description

Each input will consist of a single test case. Your program will be run multiple times on different inputs. Each input consists of two polynomials. A polynomial is given by an integer $1 \leq n \leq 20$ indicating the degree of the polynomial, followed by a sequence of integers $a_0,a_1,\ldots,a_n$, where a_i is the coefficient of x_i in the polynomial. All coefficients will fit in a signed 32-bit integer.

Output Description

For each test case, output the product of the two polynomials, in the same format as in the input (including the degree). All coefficients in the result will fit in a signed 32-bit integer.

Sample Input 1	Sample Output 1
2 1 0 5 1 0 -2	3 0 -2 0 -10

Sample Input 2	Sample Output 2
4 1 1 -1 1 1 4 9 -8 7 6 5	8 9 1 -10 30 5 -2 8 11 5

Source: Open.kattis.com (polymul1)

P –Postfix Evaluation

Problem Statement

In a postfix expression, operators follow their operands. For example, [5 2 *] is interpreted as 5 * 2. If there are multiple operators, each operator appears after its last operand. Here are more examples, showing how postfix compares to parenthesized expressions:

```
6 3 + 5 * 2 –          ((6 + 3) * 5) – 2)

4 3 2 * +              4 + (3 * 2)
```

These examples show that the operand affected by an operator can be the result of a previous calculation. Here is what you need to do: Given an integer postfix expression, you must calculate and print its value.

Input Description

Each input will consist of a single test case. Your program will be run multiple times on different inputs. A single input line will contain a postfix expression containing single digit integers and operators from the following set: { +, –, /, * } (add, subtract, divide, multiply). There will be a single space between each digit and and operator. The line will contain no more than 64 operators and numbers, total.

Output Description

Output will be a single integer on a line by itself.

Sample Input 1	Sample Output 1
`6 3 + 5 * 2 –`	`43`

Sample Input 2	Sample Output 2
`6 3 + 5 2 * *`	`90`

Sample Input 3	Sample Output 3
`4 3 2 * +`	`10`

R – Stacking Cups

Problem Statement

You are programming a cup stacking module for your robot. This robot is equiped with several sensors that can accurately determine the radius and color of a cup. Unfortunately, there is a glitch in the robot's core routine that processess sensor inputs, so the radius is doubled if the result of the color sensor arrives to the routine after the radius. For example, for a red cup with a radius of 5 units, your module will receive either "red 5" or "10 red" message.

Given a list of messages from the core routine, each describing a different cup, can you put the cups in order from smallest to largest?

Input Description

Each input will consist of a single test case. Your program will be run multiple times on different inputs. For each test case, the first line of input contains an integer N, the number of cups ($1 \leq N \leq 20$). The next N lines will contain two tokens each, either as "color radius" or "diameter color". The radius of a cup R will be a positive integer less than 1000. The color of a cup C will be a non-empty string of lowercase letters of length at most 20. All cups will be different in both size and color.

Output Description

Output the colors of the cups, one color per line, in order of increasing radius.

Sample Input	Sample Output

`3`
`red 10`
`10 blue`
`green 7`
	`blue`
	`green`
	`red`

Source: Rocky Mountain Regional Contest (RMRC) 2016