CodeWars 2007

Problem #1 -- Powers of Two

NOVICE TEAMS ONLY -- 2 Points

JAVA programmers: your program name must be: prob01.class
C programmers: your program name must be: prob01.exe

Task Description

Read a nonnegative integer N (range 0 to 20) and print the powers of 2 from 1 to 2^N. You do not need to validate that the integer is in range.

Sample Input/Output

```
Enter an integer:  6

      2^0 = 1
      2^1 = 2
      2^2 = 4
      2^3 = 8
      2^4 = 16
      2^5 = 32
      2^6 = 64
```

CodeWars 2007

Problem #2 -- Vertical Printing

NOVICE TEAMS ONLY -- 3 Points

JAVA programmers: your program name must be: prob02.class
C programmers: your program name must be: prob02.exe

Task Description

The CodeWars judges and scorekeepers need help printing the team names above each column on our score sheet. But we want them to look nice, so they need to be printed vertically. All we have to use is a list of team names. Please write us a program that will take a list of team names, and print them vertically.

Program Input

Your program will prompt for a team name. Continue prompting for names until you receive "END". Unlike the real CodeWars, this one will be limited to 16 teams maximum.

Program Output

Your program will output to the screen each team's name, in the order it was entered, arranged vertically such that it is read from top to bottom. There should be two spaces between each name as printed.

Sample Input/Output

```
Enter team name: Aardvarks
Enter team name: Emus
Enter team name: Road Runners
Enter team name: Coyotes
Enter team name: END

A  E  R  C
a  m  o  o
r  u  a  y
d  s  d  o
v        t
a     R  e
r     u  s
k     n
s     n
      e
      r
      s
```

CodeWars 2007

Problem #3 -- Combination

4 Points

JAVA programmers: your program name must be: prob03.class
C programmers: your program name must be: prob03.exe

Task Description

The combination of n things taken m at a time is a value often used in mathematics and business. With this program you will read input values, n and m, and output the number of different combinations of m things that can be formed from a group containing n members. This can be calculated by computing the factorial of n and dividing it by the factorial of m and then further dividing the result by the factorial of (n – m). The value, n-factorial is computed by multiplying n by n – 1, then multiplying the result by n – 2, and so forth, until the final result is multiplied by 1. n-factorial is sometimes written as n! Thus, 5-factorial (5!) is calculated in the following fashion: 5 x (5 – 1) x (5 – 2) x (5 – 3) x (5 – 4) = 5 x 4 x 3 x 2 x 1 = 120. 3! is calculated similarly as 3 x (3 – 1) x (3 – 2) = 3 x 2 x 1 = 6. And (5 – 3)!, otherwise known as 2!, is calculated as 2 x (2 – 1) = 2 x 1 = 2. Thus, the number of different combinations of 5 things taken 3 at a time is calculated as follows:

n! / [m! x (n – m)!]
= 5! / [3! x (5 – 3)!]
= 120 / [6 x2]
= 120 / 12
= 10.

Your program must calculate this value for any values of n and m. You must output the result as an integer. For this problem, you will not need to worry about integer overflow. All input is read from stdin. Inputs n and m will be separated by a single space and terminated by a newline character.

Sample Input/Output

```
Enter n and m:  10 5
252
```

CodeWars 2007

Problem #4 -- Password Analyzer

4 Points

JAVA programmers: your program name must be: prob04.class
C programmers: your program name must be: prob04.exe

Task Description

You're working for a growing e-commerce web site, which has become a popular target for thieves. The thieves gain access to customers' accounts by guessing passwords, which are all too often trivial (such as "secret", "password", and "1234"). If your customers used better passwords, your company would have less trouble with fraudulent purchases.

You've been tasked with creating a password analyzer, which will inform a customer about the relative strength of their choice of password. A "strong" password is one that is hard to guess – either by sheer length, or by using a combination of letters, numbers and symbols. For your assignment, a strong password has all of the following characteristics:

- Is at least 8 characters long (example: "spookyfish")
- Includes both upper and lower case letters (example: "sPookyFISH")
- Includes letters and at least one number or symbol (examples: "sPookyFISH3" or "$PookyFI3H")

A "good" password has two of these characteristics, an "acceptable" one has only one. A password that doesn't meet any of these would be considered "weak". Write a program that will analyze a given password and output the strength rating of the chosen password.

Program Input

Prompt for the password to be analyzed. You should allow for a maximum password length of 30 characters. No spaces are allowed.

Program Output

Your program will output to the screen the relative strength of the password typed, using the characteristics listed above.

Sample Input/Output

```
Enter your password: lizard
This password is WEAK

Enter your password: aardvark
This password is ACCEPTABLE

Enter your password: Aardvark
This password is GOOD

Enter your password: Aardvark77
This password is STRONG
```

CodeWars 2007

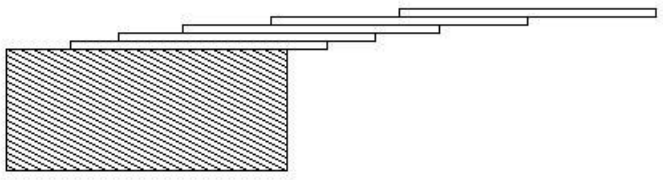
Problem #5 -- Overhanging Cards

5 Points

JAVA programmers: your program name must be: prob05.class
C programmers: your program name must be: prob05.exe

Task Description

How far can you make a stack of cards overhang the edge of a table? If you have one card, you can create a maximum overhang of half a card length. (We're assuming that the cards must be perpendicular to the table.) With two cards you can make the top card overhang the bottom one by half a card length, and the bottom one overhang the table by a third of a card length, for a total maximum overhang of 1/2 + 1/3 = 5/6 card lengths. In general you can make n cards overhang by 1/2 + 1/3 + 1/4 + ... + 1/(n + 1) card lengths, where the top card overhangs the second by 1/2, the second overhangs the third by 1/3, the third overhangs the fourth by 1/4, etc., and the bottom card overhangs the table by 1/(n + 1). This is illustrated in the figure at right.



Program Input

The input is a single line containing a positive floating-point number c (representing the desired overhang in card lengths) whose value is at least 0.01 and at most 5.20; it will contain exactly three digits.

Program Output

Output the minimum number of cards necessary to achieve an overhang of at least c card lengths. Use the exact output format shown in the examples.

Sample Input/Output

```
Enter overhang > 1.00
3 card(s)

Enter overhang > 3.71
61 card(s)

Enter overhang > 0.04
1 card(s)

Enter overhang > 5.19
273 card(s)
```

CodeWars 2007

Problem #6 -- Prime Directive

6 Points

JAVA programmers: your program name must be: prob06.class
C programmers: your program name must be: prob06.exe

Task Description

Given a positive integer N, print a list of the prime numbers from 2 to P, where P is the largest prime number ≤ N. The integer N may range from 1 to 1000. A prime number is defined as a positive integer greater than one that is exactly divisible only by itself and one. The output list should be five items per line as formatted in the example below.

Sample Input/Output

```
Enter maximum value: 50

2     3       5       7      11
13    17      19      23     29
31    37      41      43     47
```

CodeWars 2007

Problem #7 -- RAID Sizer

7 Points

JAVA programmers: your program name must be: prob07.class
C programmers: your program name must be: prob07.exe

Task Description

Servers use a number of techniques to increase hard disk performance, while also adding duplicate disks (known as redundancy) to prevent a loss of data if one fails. These techniques are known as a *Redundant Array of Inexpensive Disks* or "RAID". There are three commonly used RAID levels:

- RAID 0: Two or more identical disks are made to appear as one large disk, with all data "striped" across the disks. This increases performance, but doesn't offer any protection in case one disk fails. Example: Three 250GB disks arranged in a RAID 0 array would appear to the user as a single, 750GB disk.
- RAID 1: For each disk, a duplicate disk is added to the system, with all data "mirrored" on both disks. While this does not increase capacity, it allows for one of the disks to fail without losing data. Mirrored disks are always added in identical pairs. Example: A 500GB disk is mirrored to another 500GB disk. The user sees one 500GB disk, but can sleep better knowing either disk can crash without her data being lost.
- RAID 5: This technique stripes data across 2 or more identical disks (up to 7), and produces "parity data" on an additional identical disk for redundancy. This is like using RAID 0 and adding a single mirror disk to handle the whole group. Example: Four 400GB disks are arranged in a RAID 5 array – three of them striped, giving a capacity of 1200GB. The fourth disk is used only for fault tolerance.

Write a program to determine the cheapest array configuration. Your array can use up to 8 disks of identical capacity. Your array can use the following capacities (prices): 250GB ($75), 400GB ($110), 500GB ($140), 750GB ($250).

Program Input

Your program must prompt for the array's capacity (not including capacity used for redundancy), and the desired RAID level: 0 (zero), 1 (one) or 5. Note that the array capacity may be larger than requested, based on the disk sizes.

Program Output

The program must output to the screen the least expensive combination of disks required to build the array, including the size and quantity of disks needed, and the total user capacity and cost of the array.

Sample Input/Output

```
Enter array capacity (in GB):   450
Enter RAID level (0, 1, 5):  1

Qty: 2  Disk: 500GB  Price:  $140
Total price of this 500GB array: $280

Enter array capacity (in GB):   1200
Enter RAID level (0, 1, 5):  0

Qty: 3  Disk: 400GB  Price:  $110
Total price of this 1200GB array: $330

Enter array capacity (in GB):   600
Enter RAID level (0, 1, 5):  5

Qty: 4  Disk: 250GB  Price:  $75
Total price of this 750GB array: $300
```

Sample Input/Output

```
Enter array capacity (in GB):   1300
Enter RAID level (0, 1, 5):  1

Qty: 6  Disk: 500GB  Price:  $140
Total price of this 1500GB array: $840

Enter array capacity (in GB):   3000
Enter RAID level (0, 1, 5):  5

Qty: 7  Disk: 500GB  Price:  $140
Total price of this 3000GB array: $980

Enter array capacity (in GB):   4500
Enter RAID level (0, 1, 5):  5

Qty: 7  Disk: 750GB  Price:  $250
Total price of this 4500GB array: $1750
```

CodeWars 2007

Problem #8 -- Number Spiral

10 Points

JAVA programmers: your program name must be: prob08.class
C programmers: your program name must be: prob08.exe

Task Description

Write a program to create a number spiral of size n (n rows by n columns), where n is any odd number between 1 and 19. The spiral starts in the middle with number zero, moves outward to the right and spirals counter-clockwise.

Program Input

Prompt for the size of the number spiral. If the number is outside of the range of 1-19, or is not an odd number, your program should output an error message and exit.

Program Output

The program must output to the screen the number spiral, with the numbers aligned vertically with the least significant digits. Ensure that your largest spiral will still fit without the text wrapping on an 80 column screen.

Sample Input/Output

```
Enter the size of the number spiral: 3

    4    3    2
    5    0    1
    6    7    8

Enter the size of the number spiral: 5

   16   15   14   13   12
   17    4    3    2   11
   18    5    0    1   10
   19    6    7    8    9
   20   21   22   23   24

Enter the size of the number spiral: 7

   36   35   34   33   32   31   30
   37   16   15   14   13   12   29
   38   17    4    3    2   11   28
   39   18    5    0    1   10   27
   40   19    6    7    8    9   26
   41   20   21   22   23   24   25
   42   43   44   45   46   47   48
```

CodeWars 2007

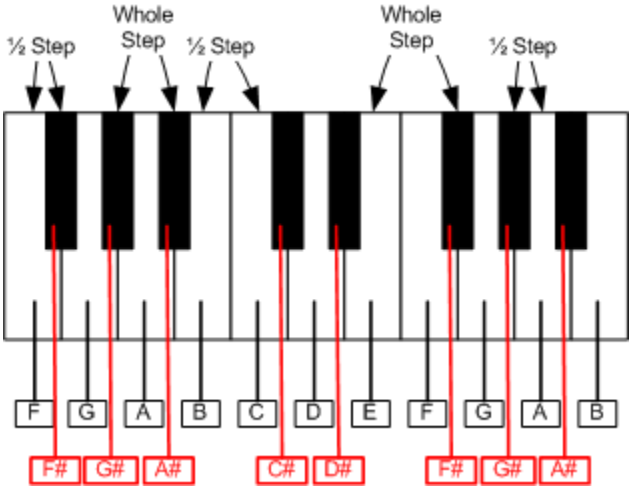
Problem #9 -- Musical Intervals

11 Points

JAVA programmers: your program name must be: prob09.class
C programmers: your program name must be: prob09.exe

Task Description

A musical interval is defined as the distance between one note on a scale and another note on the same scale. We can represent notes on a piano keyboard:



Beneath the keyboard graphic are the names of the notes, above the keyboard graphic are indications of *half steps* and *whole steps*. This keyboard graphic shows only some of the available keys. Notes drop to lower tones as you move to the left and raise to higher tones as you move to the right. The notes repeat themselves, A through G, with additional notes – called *sharps* (black keys) – in between the other notes (white keys). You move a half step when you move from one key to the immediately adjacent key, and you move a whole step when you move from one key to a key that is two keys away. Also note, for example, that while E and F (or B and C) are adjacent white keys, F and G (or G and A, or C and D, or A and B) are not.

This problem works with major scales. A major scale is that series of eight notes (called an octave) which begins and ends with the same note name and progresses from one note to the next via the following progression: 1) Initial note (e.g. A); 2) Whole step higher (B); 3) Whole step (C#); 4) Half-step (D); 5) Whole step (E); 6) Whole step (F#); 7) Whole step (G#); 8) Half step (A, again). You can begin with any key and follow this pattern to derive any of the major scales. Any other major scale can be derived by following this same pattern, beginning with a base note. For example: Initial note: G#; whole step: A#; whole step: C; half step: C#; whole step: D#; whole step: F; whole step: G; half step: G#.

Now, a specific *interval* represents the space between one note and a referenced note on the same scale, counting from the initial note as one (1) and moving along the scale to the named ordinal. Thus, a *second* interval up along the A major scale from B is C# (B=1, C#=2). A *seventh* interval down along the A major scale from G# is A (move left to go down, G#=1, F#=2, E=3, D=4, C#=5, B=6, A=7).

Your program will read a single scale name and a series of intervals (maximum of 15). The scale name can be a single letter (e.g. A) or a letter plus a sharp sign (e.g. A#). The program must walk the input intervals and output each note described along the input scale, starting with the input note and continuing to the final note. Begin each interval at the note reached by moving the previous interval. Intervals to the right are represented in the input by a plus (+) sign. Intervals to the left are represented by a minus (-) sign.

Sample Input/Output

```
Enter intervals:  A+2+5-6+9+10-11+3+3

A B F# A B D A C# E

Enter intervals:  G#-3+6-13+15+2-7+2-5+6-2+7

G# F C# F F G G# A# D# C A# G#

Enter intervals:  B-3+2-7

B G# A# B
```

CodeWars 2007

Problem #10 -- New Math

12 Points

JAVA programmers: your program name must be: Prob10.class
C programmers: your program name must be: prob10.exe

Task Description

A number system is a means of interpreting symbols used for Mathematic expression. The base of a number system is value raise to some power by which each place in the number symbol is multiplied in order to come up with the value represented by that place. For example, in base 10, the number `1234` represents the value $(1x10^3)+(2x10^2)+(3x10^1)+(4x10^0)$. The same number in base 19 represents the value $(1x19^3)+(2x19^2)+(3x19^1)+(4x19^0)$. If we let a, b, c… symbols represent the place values beyond 9 (i.e. 10, 11, 12…), then the number `aib` in base 19 represents the value $(10x19^2)+(18x19^1)+(11x19^0)$.

Your task is to interpret numbers from different number systems, execute mathematical operations on those numbers, and output the result in another number system. Each input number will be formatted as both a value and a base, with a carat (^) symbol separating them. So the number abc in base 17 will be expressed as `abc^17`. Applicable operations include addition, subtraction, and multiplication (no division). You must execute operations in the correct order, i.e. all multiplications are executed before any additions or subtractions can be executed. No parenthetical groupings will be present. The operators, +, -, and * will be used to represent addition, subtraction, and multiplication, respectively. These operators will separate input numbers, and your program must interpret them correctly. No whitespace will be present in the inputs except for a newline character at the end of the input stream.

The last data input will be an equals sign (=), followed by a carat (^) and then the base in which you must output your result.

There will be no bases smaller than base 2; there will be no bases larger than base 20. All bases larger than base ten will use the lowercase alphabetic symbols a, b, c… to represent place values larger than 9.

Sample Input/Output

```
Expression:  1b^13+j3a67^20-hhh^19+123^6*123^7=^15
408260^15

Expression: 1db7^14+egg^18-hi32^20+3876^9-321^4=^7
-1051236^7

Expression: 1db7^14+egg^18+hi32^20*3876^9*2j^19-iia62^18+321^4=^20
ia9j5bjc^20
```

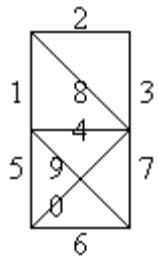
CodeWars 2007

Problem #11 -- LED Decoder

13 Points

JAVA programmers: your program name must be: Prob11.class
C programmers: your program name must be: prob11.exe

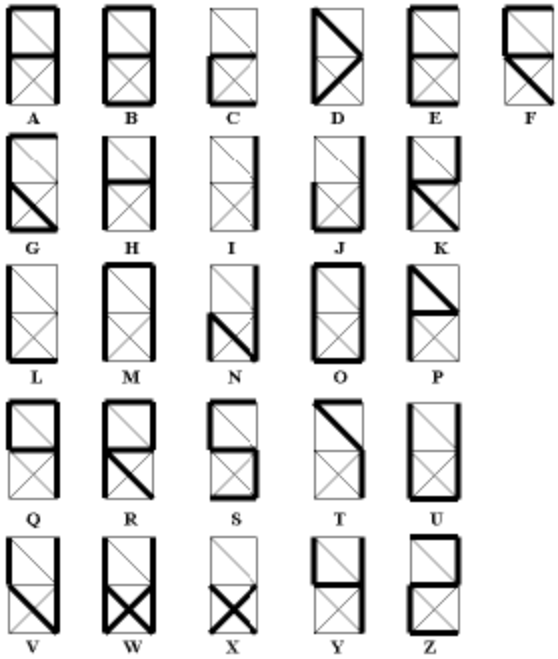
Task Description



Some LED systems display characters as a combination of light segments, much like some calculators or gas pumps do. Suppose we have such an LED system, in which each letter of the standard English alphabet is constructed by combining some of ten possible light segments, numbered as shown at right: For instance, the letter A is shown using the lines 1,2,3,4,5 and 7. With these few segments, it is not possible, of course, to show all the 26 letters with their natural shapes. A complete list of the letters of our system is shown below. Your task is to translate a combination of numbers (representing light segments) into their respective letters, finally forming a whole word or phrase. Your input is a string of letters and/or numbers. Your output will be a string of letters.

Program Input

The input is a file Prob11.in containing a single line representing an input phrase. Each input phrase consists of all uppercase letters, blank spaces and/or digits. In the case of digits, their combination must form valid LED letters. Each letter is encoded as a combination of numbers, ordered in the form 1,2,3,4,5,6,7,8,9,0. A zero (0) that is not part of a valid letter code is interpreted as a blank space. You may assume that no invalid codes are entered, and that the system does not allow ambiguity between two letter codes.



Program Output

Output the results of the decoding process. The alphabetic letters and blank spaces must not be decoded at all: only the numbers must be converted to the corresponding LED letters.

Sample Input/Output

```
Phrase: HELL1235670WO1234591561580
HELLO WORLD

Phrase: PROGRAMMING037124670C123567123567156
PROGRAMMING IS COOL

Phrase: AND MORE03712490373579027813457371246704561235673579278124561246727 8
AND MORE IF IN THIS CONTEST
```

CodeWars 2007

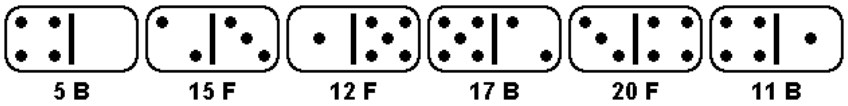
Problem #12 -- Domino Rally

14 Points

JAVA programmers: your program name must be: Prob12.class
C programmers: your program name must be: prob12.exe

Task Description

A popular diversion among domino players involves laying several of the dominoes from the set (selected at random) end-to-end, and then eliminating adjacent pairs of dominoes (from left to right) which touch at the same dot value. Consider this example with six dominoes shown below. The original draw resulted in these six dominoes.



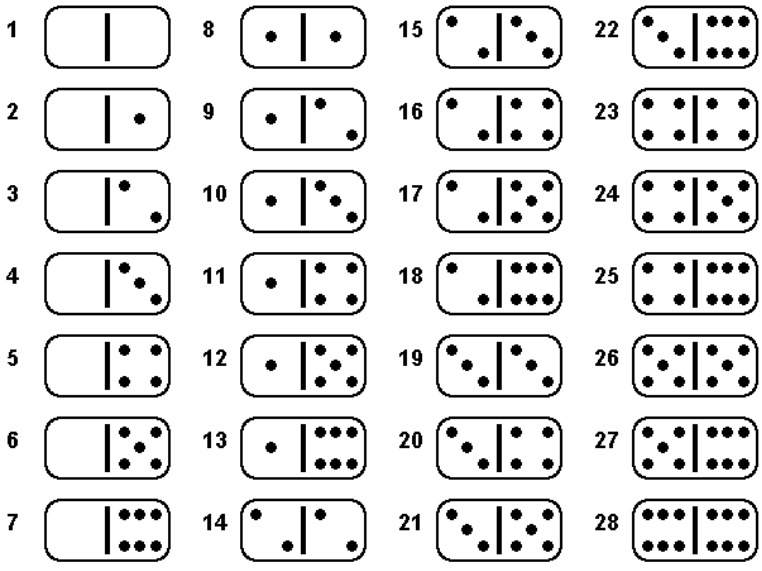
Note that when each domino is aligned, it may appear in either its forward or its backward orientation (the table above shows the forward orientations of all non-doubles). Play begins from the left end and returns to the left end after any pair of dominoes is eliminated. In this case, domino #12 (in its forward orientation) and domino #17 (in its backward orientation) are eliminated from the row since they touch each other with 5 dots.

The row is collapsed and play begins again at the left end. This time, dominoes #15 and #20 (both in their forward orientations) are eliminated, since they touch each other with 3 dots. The row is again collapsed.

Since no further eliminations can be made, the game ends. Note that dominoes #20 and #11 could not be eliminated since they were never the left-most pair.

Write a program that will accept a row of domino ID numbers and their orientations, and carry out any such successive left-most eliminations until no more are possible.

Program Input



The input file (Prob12.in) will contain three domino datasets, each corresponding to a left-to-right sequence of dominoes. A domino record consists of an integer from 1 through 28 (denoting the domino ID number, shown at right), a space, and a single character (uppercase F or B) which indicates the domino's orientation (Forward or Backward). Each domino dataset will consist of from 1 to 28 domino records, followed by a terminating record (`0 F`). Each domino record will appear on a line by itself. You may assume that each domino will appear at most once in each dataset and that all data will be valid.

Program Output

Your program will output which dominoes remain in each sequence following the elimination procedure. The report will consist of one line for each domino dataset. Each of these lines will contain the domino ID numbers which remain in each set after all eliminations have been carried out (in sequence from left to right). Domino orientations are *not* to be printed. Should it ever be possible to eliminate every domino in a dataset, the line "`DATASET CLEARED`" should be printed in place of any domino ID numbers.

Sample Input

```
5 B
15 F
12 F
17 B
20 F
11 B
0 F
18 F
22 B
0 F
23 F
12 B
2 B
4 F
15 B
20 B
19 B
7 B
6 F
0 F
```

Sample Output

```
5 11

DATASET CLEARED

19
```

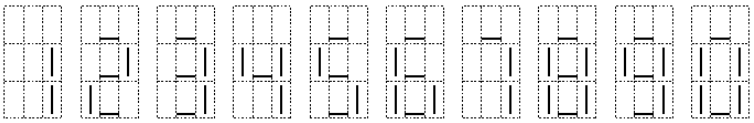
CodeWars 2007
Problem #13 -- Not Quite OCR
15 Points

JAVA programmers: your program name must be: Prob13.class
C programmers: your program name must be: prob13.exe

Task Description

Banks, always trying to increase their profit, asked their computer experts to come up with a system that can read bank checks; this would save money in check processing. One idea was to use optical character recognition (OCR) to recognize account numbers printed using 7 line segments.

Once a check has been scanned, image processing software would convert the horizontal and vertical bars to ASCII bars '|' and underscores '_'. The ASCII 7-segment versions of the ten digits look like this:



A bank account has a 9-digit account number with a checksum. For a valid account number, the following checksum equation holds: (d1 + 2 × d2 + 3 × d3 + . . . + 9 × d9) mod 11 = 0. Digits are numbered from right to left like this: d9d8d7d6d5d4d3d2d1.
Unfortunately, the scanner sometimes makes mistakes: some line segments may be missing. Your task is to write a program that deduces the original account number, assuming that:

> **Definition : mod**
> x mod y = 0 when x is divisible by y with no remainder.

- when the input represents a valid account number, it is the original number;
- at most one digit is garbled;
- garbling will NOT convert one valid digit to another (a garbled digit will never be a valid digit). Essentially, this means the check was misprinted.
- the scanned image contains no extra segments.

For example, the following input:

```
    _  _     _  _  _  _  _
  | _| _||_||_ |_   ||_||_|
  ||_  _|  | _||_|  ||_| _|
```

represents the account number "123456789".

Program Input

The input file starts with a line with one integer specifying the number of account numbers that have to be processed. Each account number occupies 3 lines of 27 characters (there are no extra spaces separating the digit representations).

Program Output

For each account number you process, your output should contain one line with 9 digits if the correct account number can be determined, the string "`failure`" if no solutions were found and "`ambiguous`" if more than one solution was found.

Sample Input

```
5
    _  _     _  _  _  _  _
  | _| _||_||_ |_   ||_||_|
  ||_  _|  | _||_|  ||_| _|

    _  _     _  _  _  _
  | _| _||_||_ |_   ||_|  |
  ||_  _|  | _||_|  ||_|  |

    _  _  _  _  _  _     _
|_||_|| || ||_   |  |  || |
  | _||_||_||_|  |  |  | _|

 _  _  _  _  _  _  _  _  _
|_||_||_||_||_||_||_||_||_|
|_||_||_||_||_||_||_||_||_|

 _  _  _  _  _  _  _  _  _
|_|  |_||_||_||_||_||_||_|
|_|  ||_||_||_||_||_||_||_|
```

Sample Output

```
123456789
failure
490067719
failure
878888888
```

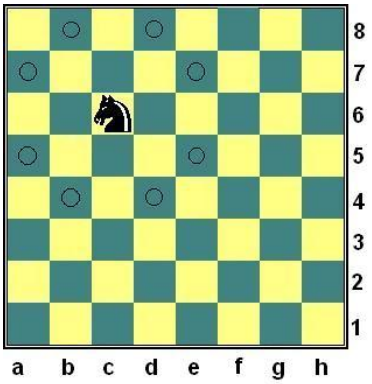
CodeWars 2007

Problem #14 -- Chess Knight

18 Points

JAVA programmers: your program name must be: Prob14.class
C programmers: your program name must be: prob14.exe

Task Description



In the game of chess, the knight moves in an interesting and unique way. It can only move in a "L"-shaped path, moving two spaces horizontally or vertically, then moving one space perpendicular to the first two. In the picture on the right, the standard 8x8 chess board is shown, with a knight in square c6. From this position, the knight can move to eight possible locations on the board, shown as circles.

Your task is write a program that will determine the shortest path a knight would take from one position on the chessboard to another. But unlike the actual game, you cannot land on squares containing other chess pieces (we'll call these "blockers"), but must go around them. Note that the knight can move over blocking pieces during its 3-square move, but must finish each move on an unoccupied space.

Program Input

Your program will prompt for the starting location of the knight, and the desired finish location, specified by the column and row. For example, the upper left corner of the chess board is location a8, and the lower right corner is h1. You may assume that all letters will be lower case.

Then, your program will prompt for one or more blocking pieces, specified by their location on the chess board. Continue prompting for more blocking pieces until you see "xx". There will be at most 31 blockers, and you can assume all input will be valid.

Program Output

Your program will output to the screen the knight's shortest path from the starting location to the finish location, with each move along the path. You must also state the total number of moves required. Note that there may be several correct (shortest) paths to the finish.

Sample Input/Output

```
The knight starts at: c6
The knight finishes at: c5
Enter blocking piece (xx to quit): b3
Enter blocking piece (xx to quit): a6
Enter blocking piece (xx to quit): b7
Enter blocking piece (xx to quit): d7
Enter blocking piece (xx to quit): xx

The knight's shortest path is 3 moves: c6  e5  d3  c5
```