**Java program:** Prob12.java

**Input File:** Prob12.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

You are on vacation in orbit around the planet Neptune. You are having so much fun that you are willing to let your inhibitions fly away with the solar wind and take a selfie to send to your relatives on Earth. Your space phone can send the image to Earth; however, since Neptune and Earth are so far away from each other, radiation interference can corrupt the data. Furthermore, in order to not tie up the space phone network, data must be sent in small chunks. Luckily for you, technology has advanced to the point that messages are received in the order they are sent. Because of the potential for interference, the space phone receiver on Earth may need to ask your space phone for parts of the image multiple times depending on interference and data corruption. A space phone company, Luca Industries, uses the Patriot Protocol to ensure message integrity.

Each Message (M) from your phone has Information (I) and a Remainder (R). Together they form what is known as the Luca Industries Data Chunk.

The Luca Industries Data Chunk:

+---------------------------------------------+

| Message |

|-------------------------------+-------------|

| Information | Remainder |

|-------------------------------|-------------|

|1 1 1 0 1 1 1 0 | 0 0 1 |

+-------------------------------+-------------+

The protocol provides a way for the message to the receiver to ensure with a high degree of certainty that there was no data corruption. This is done by adding extra digits (the remainder) to the end of the message.

The sender and receiver of any transmission using the Luca Industries Patriot Protocol use a pre-defined polynomial as part of the protocol. We call this the Patriot Protocol Polynomial, or P3 for short. For this problem, your P3 is:

1011

When sending a message from your space phone, this polynomial is used to determine the remainder to append to the information for each message. This is done using binary long division of the message by the polynomial. The length of the remainder is the length of the polynomial minus one.

**Encoding**

The following example shows how we would encrypt the data 11101110 using the Patriot Protocol:

Data Remainder

|-------------------------------|-------------|

| 1 1 1 0 1 1 1 0 | 0 0 0 |

| 1 0 1 1 | | <- P3

| --------------------------------------------|

| 0 1 0 1 1 1 1 0 | 0 0 0 |

| 1 0 1 1 | | <- P3

| --------------------------------------------|

| 0 0 0 0 0 1 1 0 | 0 0 0 |

| 1 0 1 | 1 | <- P3

| --------------------------------------------|

| 0 0 0 0 0 0 1 1 | 1 0 0 |

| 1 0 | 1 1 | <- P3

| --------------------------------------------|

| 0 0 0 0 0 0 0 1 | 0 1 0 |

| 1 | 0 1 1 | <- P3

| --------------------------------------------|

| 0 0 0 0 0 0 0 0 | 0 0 1 | <- Remainder

|-------------------------------|-------------|

The remainder 001 is then appended to the message yielding the Luca Industries Data Chunk 11101110001

**Decoding**

The receiving side performs the same binary long division. If the data integrity was maintained, there should be no remainder after division. Otherwise, we can assume there was data corruption during transmission.

Data Remainder

|-------------------------------|-------------|

| 1 1 1 0 1 1 1 0 | 0 0 1 |

| 1 0 1 1 | | <- P3

| --------------------------------------------|

| 0 1 0 1 1 1 1 0 | 0 0 1 |

| 1 0 1 1 | | <- P3

| --------------------------------------------|

| 0 0 0 0 0 1 1 0 | 0 0 1 |

| 1 0 1 | 1 | <- P3

| --------------------------------------------|

| 0 0 0 0 0 0 1 1 | 1 0 1 |

| 1 0 | 1 1 | <- P3

| --------------------------------------------|

| 0 0 0 0 0 0 0 1 | 0 1 1 |

| 1 | 0 1 1 | <- P3

| --------------------------------------------|

| 0 0 0 0 0 0 0 0 | 0 0 0 | <- All zeroes!

|-------------------------------|-------------|

Notice that the P3 always slides to a position where its leftmost bit is beneath the leftmost bit in the data that contains a 1. Also notice that the division is done using the exclusive or function on the bits of the data and the P3 (meaning a 1 results if either the P3 or the data contains a 1, but not if they both do).

**Program Input**

The first line of the file Prob12.in.txt will contain a positive integer T denoting the number of test cases that follow. Each test case will have the following input:

* A single 11 digit pre-encoded Luca Industries Data Chunk

**Example Input:**

5

11001101110

10000111010

10101011110

10000110111

11001111000

**Program Output**

For each test case, your program should either output “ok” if the data was not found to be corrupt, or “corrupt” if it was.

**Example Output:**

ok

corrupt

corrupt

corrupt

ok

**Java program:** Prob13.java

**Input File:** Prob13.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

The commutative property of addition means that if you are adding numbers together, it doesn’t matter what order you add them in. 1 + 2 = 3, and 2 + 1 = 3. You will use this special property of addition to show just how many different ways you can sum up numbers. Given a list of numbers, you must write a program that finds all possible multiset permutations of these numbers that add up to a specified sum.

What is a multiset permutation? I’m glad you asked. It is a permutation of a set of objects (in this case digits) where objects of the same type are freely interchangeable. In the example input below, the second test case has two ‘1’ digits. However, the equation 1+1+2 only appears once in the output. That is because switching the 1s around does not form a new equation, even though they are different instances of the number 1.

**Program Input**

The first line of the file Prob13.in.txt will contain a positive integer T denoting the number of test cases that follow. Each test case will have the following input:

* The first line of each test case will be the sum you need to find in the following format:

FIND SUM=<positive integer>

* The second line of each test case will contain a list of positive integers delimited by a comma.

**Example Input:**

2

FIND SUM=10

2,3,7,1,8

FIND SUM=4

2,1,3,1

**Program Output**

Your program should output the all the possible addition equations you can make from the given numbers for the given sum. You cannot repeat any of the given numbers in your solutions unless the number is repeated in the input list. Your equations should be ordered in an ascending manner by the first number in the equation, then by the second, and so on.

**Example Output:**

1+2+7

1+7+2

2+1+7

2+7+1

2+8

3+7

7+1+2

7+2+1

7+3

8+2

1+1+2

1+2+1

1+3

2+1+1

3+1

**Java program:** Prob14.java

**Input File:** Prob14.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

This problem isn’t about Thanksgiving, it’s about bowling! If you’ve ever gone bowling, then you know that the scoring can be complicated. That’s why we want you to write a program to do it. Just in case you don’t know, here is how a bowling game works:

**Frames**

In bowling, each time the pins are set up for you to knock them down is called a frame. There are ten frames in a bowling game. You get two chances to knock down all ten pins. After your second chance, the remaining pins are discarded, and a fresh set of pins is set up to start the next frame. The only exception to this is if you get a strike or a spare in the 10th frame. If that happens, the 10th frame will have three chances due to the scoring for strikes and spares (see below).

**The Scorecard**

When you roll your bowling ball towards the pins, one of 4 things will happen:

* If you hit no pins, the scoreboard will show a dash (-). You won’t see any zeroes on a bowling scorecard.
* If you knock down all the pins on your first attempt, it is called a strike. A strike is denoted by a capital x (X).
* If you knock down all the remaining pins on your second attempt, it is called a spare. A spare is denoted by a forward slash (/).
* If you knock down some pins but some still remain, you will see the number of pins you knocked down with that ball.

**Scoring**

In general, every pin that you knock down gives you one point. There are two exceptions:

* When you get a spare, you are awarded points for the pins you knocked down to get the spare plus the number of pins you knock down with your next ball.
  + For example, if in the first frame of the game you knocked down 4 pins and then 6 pins to get a spare, and in the second frame you knocked down 3 pins with your first ball, the total score for the first frame would be 13 points (4 + 6 + 3).
* When you get a strike, you are awarded points for the pins you knocked down to get the strike (always 10) plus the number of pins you knock down with your next two balls.
  + For example, if in the first frame of the game you knock down all 10 pins with your first ball to get a strike, and in the second frame you knock down 4 pins and then 3 pins, the total score for the first frame would be 17 points (10 + 4 + 3).
  + Another example: if in the first three frames you get a strike (three strikes in a row is called a turkey, by the way), then the total score in the first frame would be 30 (10 + 10 + 10). For this reason, the maximum bowling score is 300 (30 points for each of the 10 frames).

**The 10th Frame**

The 10th frame is special because it is the last frame. Here is how the 10th frame works:

* If you do not get a strike or a spare in the 10th frame, then it is scored just like any other frame.
* If you get a spare, you are awarded one more ball to bowl (so you can add those points to the spare). If you get a strike, no more attempts are awarded. The 10 pins you knocked down with your extra ball are added to the spare, and the game is over.
* If you get a strike, you are awarded two more balls to bowl (so you can add those points to the strike). If you get a strike or a spare with your two extra balls, no more attempts are awarded. The pins you knocked down with your extra balls (20 max) are added to the strike, and the game is over.

Whew! Are you ready? Let's go bowling!

**Program Input**

The first line of the file Prob14.in.txt will contain a positive integer T denoting the number of test cases that follow. Each test case will have the following input:

* Each line will contain a single bowling scorecard (10 frames of scores). Frames will be separated by commas. There will be no incomplete games. The only characters besides commas you will encounter will be:
  + Integers 1-9
  + X (strike)
  + / (spare)
  + - (no pins knocked down)

**Example Input:**

3

--,--,--,--,--,--,--,--,--,--

X,X,X,X,X,X,X,X,X,XXX

X,13,X,81,5/,X,18,33,X,X36

**Program Output**

For each test case, your program should print out the score of the bowling game.

**Example Output:**

0

300

142