University of Scranton
ACM Student Chapter / Computing Sciences Department
15th Annual High School Programming Contest (2005)

Problem 1: Translating Three-digit Numbers into Words

Develop a program that, given as input an integer in the range 0..999, produces as output the same number, but expressed as a string of words in English. For example, the number 347 should be translated by the program into the string

```
Three Hundred Forty Seven
```

You may use all lower case (or upper case) letters if you prefer.

Input: The first line contains a positive integer n indicating how many instances of the problem are to be solved. On each of the next n lines there is a natural number (expressed in the usual decimal notation) in the range 0..999, without any leading zeros.

Output: For each of the n numbers given, the program should produce a single line of output containing the number in decimal notation, followed by a colon, followed by a space, followed by its translation into the corresponding sequence of English words, each one separated from the next by a single space.

Sample input	Corresponding output
6	0: Zero
0	37: Thirty Seven
37	408: Four Hundred Eight
408	126: One Hundred Twenty Six
126	610: Six Hundred Ten
610	18: Eighteen
18	

University of Scranton
ACM Student Chapter / Computing Sciences Department
15th Annual High School Programming Contest (2005)

Problem 2: Associativity

As you know, addition is *associative*; that is, for any numbers x, y, and z, $(x + y) + z = x+(y+z)$. Several other well known operations also have this property, including multiplication. Others do not, including subtraction.

Let $\oplus$ be a binary operation over the set S, which is to say that, for any $x, y \in S$, the expression $x \oplus y$ yields a member of S. Then $\oplus$ is said to be associative if, for every $x, y, z \in S$, $(x \oplus y) \oplus z = x \oplus (y \oplus z)$.

Develop a program that, given a tabular description of a binary operation $\oplus$ over a finite set S, reports whether or not $\oplus$ is associative. You may assume that S is comprised of the letters in some initial segment of the sequence $\langle a, b, c, \cdots, z\rangle$.

Input: The first line contains a positive integer n indicating how many instances of the problem will be described on subsequent lines. Each such instance is given by a positive integer m, $1 \leq m \leq 26$, (the size of S) appearing by itself on a line, followed on the next m lines by an $m \times m$ table defining the operation $\oplus$. Adjacent items in a row are separated from one another by a single space.

For example (assuming that $m \geq 7$), the value of $g \oplus c$ is found in the third column (because c is the third letter in the alphabet) of the seventh row (because g is the seventh letter).

Output: For each instance of the problem appearing in the input, your program should display either "associative" or "not associative", whichever is appropriate. In the latter case, your program should display, in addition, a sequence of three elements of S "giving witness" to the fact that the associativity property is not met. For example, the sample output below indicates that $(c \oplus c) \oplus b \neq c \oplus (c \oplus b)$ in the second instance described in the sample input.

```
Sample input
------------
2
4
a b c d
b c d a
c d a b
d a b c
3
a a a
b b a
b a c
```

```
Corresponding output
--------------------
associative
not associative: c c b
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
15th Annual High School Programming Contest (2005)

Problem 3: Mean, Median, and Mode

Develop a program that, given as input a (non-empty) collection of integers in the range 0..99, calculates its mean, median(s), and mode(s).

The *arithmetic mean* (or what is often referred to as the *average*) of a collection of values is the sum of the items divided by their number.

Here we define a *median* of a collection to be any value x in the collection having the property that at most half of the items in the collection are less than x and at most half of the items in the collection are greater than x. Note that any non-empty collection of numbers has a median; non-empty collections containing an even number of items may have two medians.

Any item in a collection that appears at least as often as any other is a *mode* of the collection.

As an example, consider the collection of integers

$$34,\ 8,\ 15,\ 8,\ 55,\ 17,\ 8,\ 20,\ 15,\ 34$$

The sum of these ten items is 214, which yields a mean of $\frac{214}{10}$, or 21.4. The two medians are 15 (three items are less, five are greater) and 17 (five items are less, four are greater). The lone mode, 8, occurs three times.

Input: The first line contains a positive integer n indicating how many collections are to be processed. Each collection is described by a postive integer m appearing alone on a line and indicating the number of items in the collection, followed by the items themselves, which appear on the following m lines, one per line. Note that there is no *a priori* upper bound on the value of m.

Output: For each collection given as input, four lines of output are to be generated, the first of which reports the collection's arithmetic mean (a real number with at least one digit after the decimal point), the second of which identifies the collection's one or two medians, the third of which identifies the collection's mode(s) and indicates how many times each of them appears, and the fourth of which is blank. (See sample output below for the details of formatting.) The medians and modes should be listed in ascending order.

Hint: Take advantage of the assumption that the items appearing in a collection are restricted to integers in the range 0..99. Absent this restriction, the problem would be somewhat more difficult to solve.

Sample Input	Corresponding Output
3 7 14 56 2 14 8 0 5 6 0 68 68 10 0 24 3 1 2 0	Mean: 14.14 Median(s): 8 Mode(s): 14 (occurs 2 times) Mean: 28.33 Median(s): 10 24 Mode(s): 0 68 (occurs 2 times) Mean: 1.0 Median(s): 1 Mode(s): 0 1 2 (occurs 1 times)

University of Scranton
ACM Student Chapter / Computing Sciences Department
15th Annual High School Programming Contest (2005)

Problem 4: That Delicate Balance

Suppose that you have a balance scale and, for each i satisfying $0 \leq i \leq 19$, a rock weighing 3^i grams. (That is, you have twenty rocks and they weigh, respectively, 1 gram, 3 grams, 9 grams, 27 grams, 81 grams, ..., and 3^{19} grams.)

Develop a program that, given as input a positive integer m that indicates the weight, in grams, of an object lying in the "left" basket of the scale, reports which rocks must be placed in the left and right baskets, respectively, in order for the two baskets to contain items of equal total weight.

Hint: For $k \geq 0$, define

$$f(k) \;=\; \sum_{0 \leq i < k} 3^i \;=\; 3^0 + 3^1 + \cdots + 3^{k-1}$$

Let L and R, respectively, be the sums of the weights of the items currently in the left and right, respectively, baskets of the scale. Let $w = L - R$. (Prior to any rocks being placed into baskets, $w = m$.) If $w = 0$, no more rocks are needed. If, on the other hand, $w \neq 0$, you should place into the "lighter" basket the rock weighing 3^j grams, where $f(j) < |w| \leq f(j+1)$, and continue from there.

Input: The first line contains a positive integer n indicating how many objects are to be weighed. Each of the n lines thereafter contains a positive integer m for which there exists a solution using the twenty rocks you are assumed to have. (To be precise, $0 < m < \frac{3^{20}}{2}$.)

Output: For each of the n objects given as input, report its weight m and the weights of the rocks that must be placed in the left and right baskets of the scale so that the two baskets contain items of equal total weight. To do this, display m followed by the weights of all the rocks placed into the baskets, in decreasing order, with weights of rocks placed in the left basket being shown as positive numbers and weights of rocks placed in the right basket being shown as negative numbers. (Thus, the sum of all the numbers displayed should be 0.)

Sample Input	Corresponding Output
3	17 -27 9 1
17	52 -81 27 3 -1
52	3462 -6561 2187 729 243 -81 27 -9 3
3462	

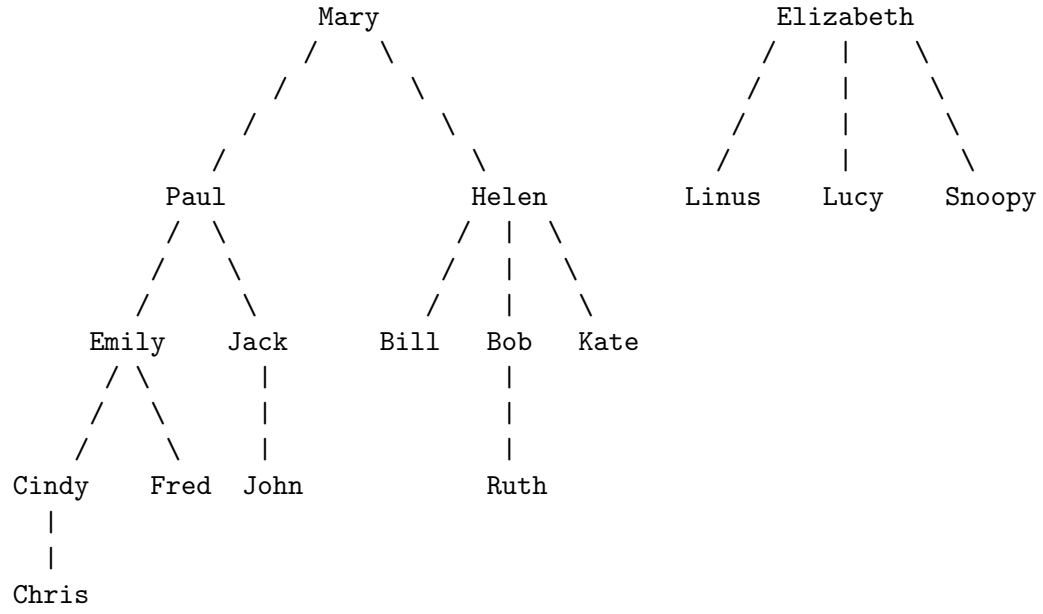
University of Scranton
ACM Student Chapter / Computing Sciences Department
15th Annual High School Programming Contest (2005)

Problem 5: All in the Family

In mathematics, a *tree* is a structure containing *nodes* and *edges*. Each edge connects two nodes, indicating that some kind of relationship holds between the two. One well known application of this concept is the *family* (or *genealogical*) tree, which shows how members of a family are related to one another.

Specifically, each node in a family tree represents a particular entity (e.g., a person) and each edge connects a child to its parent. Each node in a tree has one parent, except for the *root*, which has none. By convention, in drawing a family tree we place a parent above its children.

Consider this "forest" of two family trees:



What this depicts is that Mary has as children Paul and Helen; Elizabeth has as children Linus, Lucy, and Snoopy; Paul has as children Emily and Jack; etc., etc.

If A and B are nodes in distinct family trees, we say that A and B are *unrelated*. In our example, Linus and Emily are unrelated.

Now suppose that A and B are nodes in the same family tree. If there is a path going upward from B to A, we say that A is an *ancestor* of B (equivalently, B is a *descendant* of A). More precisely, letting k be the length of that path, we say that A is the k-ancestor of B (equivalently, B is a k-descendant of A). In our example, Mary is the 3-ancestor of John, and Jack is the 0-ancestor of himself. (Yes, we consider each person to be his or her own 0-ancestor!)

If A and B are nodes in the same family tree but neither is an ancestor of the other, they are *cousins*. More precisely, for some natural numbers p and q they are *p-th cousins q times removed*. To calculate p and q, begin at either A or B and follow the path upward until you reach a node C that is an ancestor to both of them. C is called the *nearest common ancestor* of A and B. Let m and n be such that C is the m-ancestor of A and the n-ancestor of B. Then $p = \min(m, n) - 1$ and $q = |m - n|$.

In our example, Chris and Ruth are 2nd cousins once removed. (Taking A to be Chris and B to be Ruth, we have C being Mary, $m = 4$, and $n = 3$, from which we get $p = \min(4, 3) - 1 = 3 - 1 = 2$ and $q = |4 - 3| = |1| = 1$.) Paul and Ruth are 0th cousins twice removed, while Chris and John are 1st cousins once removed.

Develop a program that, given a forest of one or more family trees, is able to describe the relationship, if any, between any two nodes in that forest. For the sake of simplicity, each node in the forest will be identified by an integer in the range 0..99 (rather than by a character string such as "Chris").

Hint: For this problem, an effective approach for representing a tree is to maintain an array `parentOf[ ]` such that, if k is the parent of j, then `parentOf`$[j] = k$.

Input: The first line contains a natural number n indicating how many edges are in the forest. On each of the next n lines there is a pair of integers x and y in the range 0..99, separated by a space, indicating that node x is a child of node y (i.e., there is an edge going upward from x to y). Because no node has multiple parents, no number will appear first on more than one of the n lines just described. Any node not mentioned at all on these lines is the root of a one-node tree.

Following the last line describing an edge will be a line containing a single natural number k. On each of the k lines that follow will be a pair of integers u and v (in the range 0..99) the relationship between which is to be computed.

Note that the sample input below corresponds to the forest depicted above. The vertices are numbered as though we visited the roots, then their children, then their grandchildren, etc., going left to right across each level.

Output: For each pair of integers u and v appearing on one of the last k lines of input, your program should produce a single line of output indicating the relationship between u and v, from the point of view of u. There are four kinds of relationships: none, ancestor, descendant, and cousin. The precise format in which each of these should be expressed is shown in the sample output.

Sample input	Corresponding output
15	4 and 7 are unrelated
4 1	0 is the 3-ancestor of 14
14 8	14 is a 3-descendant of 0
6 1	8 is the 0-ancestor of 8
13 7	16 and 15 are 2-th cousins 1 times removed
12 7	2 and 15 are 0-th cousins 2 times removed
15 10	16 and 14 are 1-th cousins 1 times removed
9 3	35 and 6 are unrelated
5 1	
7 2	
16 12	
2 0	
11 3	
3 0	
10 3	
8 2	
8	
4 7	
0 14	
14 0	
8 8	
16 15	
2 15	
16 14	
35 6	

University of Scranton
ACM Student Chapter / Computing Sciences Department
15th Annual High School Programming Contest (2005)

Problem 6: Recognizing Convex Polygons

A *polygon* is defined to be a plane figure formed by line segments such that

(1) each line segment intersects exactly two others, one at each endpoint, and
(2) no two line segments with a common endpoint are collinear (i.e., lie on the same line).

The endpoints of the line segments are referred to as the polygon's *vertices* (singular: *vertex*) and the line segments are referred to as its *sides*.

A polygon is said to be *convex* if, for any two vertices u and v, the line segment connecting them contains no points exterior to the polygon.

Below are two examples of polygons. The one on the left is convex; the one on the right is not, as is demonstrated by the dashed line segment, which connects two vertices and contains points exterior to the figure.

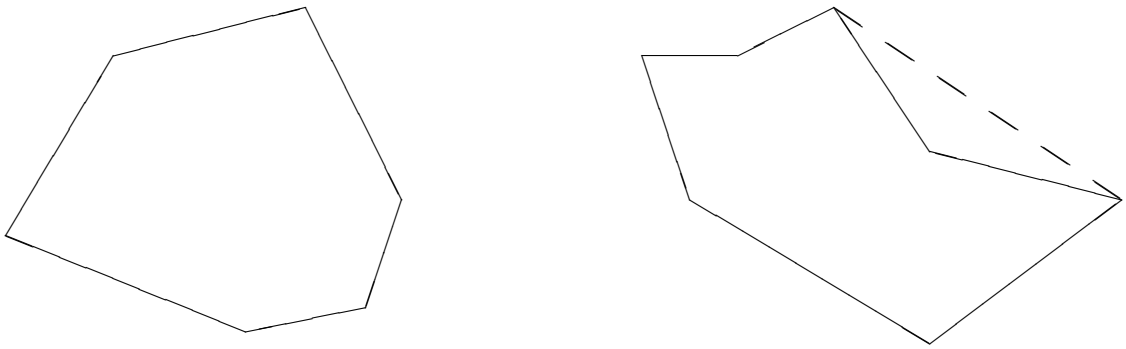


Figure 1: A Convex Polygon and a Non-convex Polygon

A standard way of representing a polygon is as a sequence $\langle p_1, p_2, \cdots, p_m \rangle$ of vertices, where, for each i satisfying $1 \leq i < m$, line segment $\overline{p_i p_{i+1}}$ forms a side of the polygon, as does $\overline{p_m p_1}$.

Imagine beginning at a point on one of the sides of a polygon and "walking" along its sides, eventually returning to the starting point. Each time you reach the endpoint of a side (i.e., a vertex), you must turn either clockwise or counterclockwise before walking along the next side.

A necessary and sufficient condition for a polygon to be convex is that either

(1) every such turn is in the clockwise direction, or
(2) every such turn is in the counterclockwise direction.

Develop a program that, given a sequence of vertices describing a polygon, reports whether or not it is convex.

Hint: Suppose that p, q, and r are points and that we walk along the line segment $\overline{pq}$ (from p to q) followed by the line segment $\overline{qr}$ (from q to r). In which direction (clockwise or counterclockwise) did we turn upon reaching q?

It turns out that it is remarkably easy to answer this question, once we have set up the appropriate machinery. Toward this end, let $A = (x_1, y_1)$ and $B = (x_2, y_2)$ be points. Then we define $A - B = (x_1 - x_2, y_1 - y_2)$ and $A \times B = x_1 \cdot y_2 - x_2 \cdot y_1$

If $(r - p) \times (q - p)$ is positive, the turn made at q is in the clockwise direction; if this value is negative, the turn is in the counterclockwise direction. *End of Hint*

Input: The first line contains a positive integer n indicating how many polygons are to be tested. Each polygon is described beginning with a positive integer $m > 2$ indicating how many vertices are in the polygon, followed by a sequence of vertices, one per line. Each vertex is given by two real numbers representing, respectively, its x- and y-coordinates.

Output: For each polygon provided as input, the program should report whether or not it is convex.

Sample input	Corresponding output
2	not convex
6	convex
-0.5 0.0	
0.8 -1.0	
2.0 1.0	
3.5 0.9	
1.5 2.5	
0.9 1.0	
4	
2.3 1.0	
1.5 -1.0	
-1.4 0.0	
1.0 2.0	