University of Scranton
ACM Student Chapter / Computing Sciences Department
17th Annual High School Programming Contest (2007)

Problem 1: Rectangle Area

Develop a program that, given as input the cartesian coordinates of three vertices of a rectangle, reports the area of that rectangle. You will recall that the area of a rectangle is the product of the lengths of any two adjacent sides.

Input: The first line contains a positive integer n indicating how many rectangles are to be analyzed. Each rectangle is described on a single line via six real numbers, x_1, y_1, x_2, y_2, x_3, and y_3, separated by spaces. These provide the coordinates of three of the rectangle's vertices, namely $P_1(x_1, y_1)$, $P_2(x_2, y_2)$, and $P_3(x_3, y_3)$.

Output: For each rectangle provided as input, report its area.

```
Sample input
------------
3
0.0 0.0 0.0 1.0 1.0 0.0
-1.0 2.0 3.0 5.0 1.0 1.0
5.0 9.0 -0.5 0.0 7.5 5.0

Resultant output
----------------
Area of rectangle with vertices (0.0,0.0),(0.0,1.0),(1.0,0.0) is 1.0
Area of rectangle with vertices (-1.0,2.0),(3.0,5.0),(1.0,1.0) is 10.0
Area of rectangle with vertices (5.0,9.0),(-0.5,0.0),(7.5,5.0) is 44.5
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
17th Annual High School Programming Contest (2007)

Problem 2: Converting Egyptian Fractions to Fractions

A *unit fraction* is a fraction of the form $\frac{1}{r}$, where r is a (non-zero) integer. An *Egyptian fraction* is a sum of distinct positive unit fractions, so called because this is the manner in which ancient Egyptians expressed fractions in general. For example, they would have written $\frac{3}{5}$ as $\frac{1}{2} + \frac{1}{10}$ (except that they would have used hieroglyphics rather than Arabic numerals).

Develop a program that, given an Egyptian fraction (i.e., a set of distinct positive unit fractions), calculates the sum and expresses it in simplest (i.e., reduced) form.

Input: The first line contains a positive integer n indicating how many instances of the problem are subsequently described. Each such instance is described on two lines, the first of which contains a single positive integer m indicating how many unit fractions are to be summed. On the second line appears a sequence of m distinct positive integers $k_1, k_2, \ldots, k_m$, which is to be interpreted as representing the Egyptian fraction $\frac{1}{k_1} + \frac{1}{k_2} + \cdots + \frac{1}{k_m}$.

Output: For each Egyptian fraction given as input, your program should generate a single line of output indicating the unit fractions that were summed (separated by plus signs), followed by an equals sign, followed by the sum, in simplest form.

Sample input	Resultant output
3	1/2 + 1/10 = 3/5
2	1/35 = 1/35
2 10	1/98 + 1/1 + 1/2 + 1/12 = 937/588
1	
35	
4	
98 1 2 12	

University of Scranton
ACM Student Chapter / Computing Sciences Department
17th Annual High School Programming Contest (2007)

Problem 3: Palindrome Trawler

A *palindrome* is a character string that reads the same forwards as backwards. Examples include *a*, *aa*, *abba*, and *fbccaccbf*.

Develop a program that, given a character string, identifies all its substrings of length three or more that are palindromes.

Input: The first line contains a positive integer n indicating the number of strings that are to be analyzed. Each such string is described on two lines, the first of which gives its length and the second of which contains the string itself.

Output: For each string given as input, echo that string on one line and then, on subsequent lines, list all its substrings of length three or more that are palindromes, one per line. Preceding each such palindrome, display its starting position within the input string. (Assume that positions are numbered beginning at 1.) The palindromes should appear in ascending order by starting position. For multiple palindromes having the same starting position, list them in ascending order by length. Following the last palindrome, display a blank line.

```
Sample Input
------------
3
7
baabbab
5
bbacb
10
aababababaab
```

```
Resultant Output
----------------
baabbab
1 baab
3 abba
5 bab

bbacb

aababababaab
1 aababababaa
2 aba
2 ababa
2 abababa
3 bab
3 babab
4 aba
4 ababa
5 bab
6 aba
7 baab
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
17th Annual High School Programming Contest (2007)

Problem 4: Run-length Decoding

Run-length encoding is a data compression technique that works well on data in which values tend to repeat frequently. Such data is common in some kinds of applications, including digital imaging. Fax machines, for example, employ this kind of compression, as does the JPEG image encoding standard.

Within a sequence, a *run* is a maximal subsequence all of whose members have the same value. (By "maximal" is meant that a run cannot be extended to include the element preceding it or the one following it, because they have values that are different from the (common) value of the run's members.)

By this definition, any sequence can be split up (uniquely) into runs. Consider, for example, this sequence S of integers:

$$S = 37\ 0\ 0\ 5\ 5\ 5\ 5\ 5\ 5\ 429\ 429\ 0\ 0\ 0\ 8\ 2\ 2\ 2\ 0\ 86\ 86\ 86\ 86\ 7\ 5\ 1\ 1\ 2$$

Using parentheses to partition it into its runs, we get

$$(37)(0\ 0)(5\ 5\ 5\ 5\ 5\ 5)(429\ 429)(0\ 0\ 0)(8)(2\ 2\ 2)(0)(86\ 86\ 86\ 86)(7)(5)(1\ 1)(2)$$

The rules for performing run-length encoding (of a sequence of integers) are as follows:

(1) Any run of length three or less whose members are non-zero is left unaltered.

(2) Any run of length four or more whose members are non-zero is encoded by three numbers: zero (which signals that a run is being encoded), followed by the length of the run, followed by the value of each member. For example, a run of 5's having length twelve is encoded as 0 12 5.

(3) Any run of length two or more whose members have value zero is encoded as in (2). For example, a run of 0's having length three is encoded as 0 3 0.

(4) A run of length one whose member is zero is encoded as 0 0.

Following these rules, the sequence S from above is encoded as

$$T = 37\ 0\ 2\ 0\ 0\ 6\ 5\ 429\ 429\ 0\ 3\ 0\ 8\ 2\ 2\ 2\ 0\ 0\ 0\ 4\ 86\ 7\ 5\ 1\ 1\ 2$$

For this particular example, in which the runs are all fairly short, very little compression is achieved. (S contains 28 values, and T contains almost as many, 26.)

Develop a program that does run-length **decoding**, by which we mean the inverse of run-length encoding. That is, given as input the result of applying run-length encoding to some sequence, the program should produce as output that sequence. For example, given T as input, the program should produce S as output.

Input: The first line contains a positive integer n indicating how many sequences are to be decoded. The next $2n$ lines contain descriptions of the sequences, with each description occupying two lines. The first line of each description contains a positive integer m (no greater than 100) equal to the length of (i.e., number of elements in) the sequence. The second line contains m nonnegative integers comprising the sequence, separated by spaces. You may assume that each sequence given as input is the valid run-length encoding of some sequence.

Output: For each sequence given as input, echo it on one line, display the result of run-length decoding it on the next line, and make the next line blank.

```
Sample Input
------------
2
10
0 6 4 0 0 71 71 0 4 5
14
1 0 2 0 0 10 1 2 3 4 0 3 0 2


Resultant output
----------------
0 6 4 0 0 71 71 0 4 5
4 4 4 4 4 4 0 71 71 5 5 5 5

1 0 2 0 0 10 1 2 3 4 0 3 0 2
1 0 0 1 1 1 1 1 1 1 1 1 2 3 4 0 0 0 2
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
17th Annual High School Programming Contest (2007)

Problem 5: Sudoku Verification

A Sudoku board is a 9×9 matrix, embedded in which are nine 3×3 sub-matrices, each cell of which is either empty or contains one of the digits 1 through 9. A Sudoku board is said to be *viable* if no digit occurs more than once in any row, column, or sub-matrix. A Sudoku board is said to be *complete* if it is viable and none of its cells is empty. (Hence, in a complete Sudoku board each of the digits 1 through 9 appears exactly once in each row, column, and sub-matrix.)

Using the terms introduced above, every Sudoku board can be classified as either **complete**, **incomplete but viable**, or **non-viable**.

Develop a program that, given a Sudoku board, reports in which of these three categories it lies. In the case of a non-viable board, the program also should identify every row, column, and sub-matrix in which there is a violation of viability (i.e., in which some digit occurs more than once).

Input: The first line contains a positive integer n indicating how many Sudoku boards are to be classified. Each board is described on nine lines, one row per line. Using zero to indicate an empty cell, each row is described by the nine digits in its cells, separated from one another by spaces. A blank line separates each board from the next.

Output: For each board given as input, generate a single line of output that properly classifies it. Specifically, that line should contain one of the strings `complete`, `incomplete but viable`, or `non-viable`. In the case of a non-viable board, three more lines of output should be generated, the first of which lists any rows violating viability, the second of which lists any columns violating viability, and the third of which lists any sub-matrices violating viability. (See sample output for correct formatting.) Rows are numbered 1 through 9 going top to bottom, columns are numbered 1 through 9 going left to right, and sub-matrices are numbered 1 through 9 according to the following picture, in which each box represents a 3×3 sub-matrix.

```
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | 9 |
+---+---+---+
```

```
Sample input
------------
3
1 2 3 4 5 6 7 8 9
4 5 6 7 8 9 1 2 3
7 8 9 1 2 3 4 5 6
2 3 4 5 6 7 8 9 1
5 6 7 8 9 1 2 3 4
8 9 1 2 3 4 5 6 7
3 4 5 6 7 8 9 1 2
6 7 8 9 1 2 3 4 5
9 1 2 3 4 5 6 7 8

1 2 0 4 0 6 7 0 9
0 5 6 7 0 9 0 2 3
7 8 9 1 2 3 4 5 6
0 0 4 0 6 0 0 0 1
5 0 7 0 9 0 2 3 0
0 0 1 0 3 0 0 6 7
3 0 5 0 7 0 0 0 0
0 0 0 9 0 0 0 4 5
9 1 0 0 4 0 6 0 8

1 2 0 4 0 6 7 0 9
0 1 6 7 0 9 0 2 3
7 8 9 1 2 3 4 5 6
0 0 4 0 6 0 0 7 1
5 0 7 0 9 0 2 3 0
0 0 1 0 3 0 0 6 7
3 0 5 0 4 0 0 0 0
0 0 0 9 0 0 0 4 5
9 1 0 0 4 0 6 0 8
```

```
Resultant output
----------------
complete
incomplete but viable
non-viable
  rows:
  columns: 2 5
  sub-matrices: 1 6 8
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
17th Annual High School Programming Contest (2007)

Problem 6: Counting Points in Basketball

In basketball, a free throw counts as a single point, a (normal) field goal counts as two points, and a 3-point field goal counts (surprise!) as three points. Suppose that, as a game progresses, we keep track of how a team scored its points by writing either 1, 2, or 3 each time it scores, according to how it scored. For example, if a team scores via three normal field goals, followed by two free throws, followed by another normal field goal, and then by a 3-point field goal, we will have written the sequence $\langle 2, 2, 2, 1, 1, 2, 3 \rangle$. This is just one of many possible sequences describing how a team could have scored 13 points. Another one is $\langle 2, 2, 1, 3, 2, 2, 1 \rangle$. Notice that, even though these two sequences contain the same collection of values (two occurrences of 1, four occurrences of 2, and a single occurrence of 3), they are distinct because their elements appear in different orders.

Develop a program that, given a nonnegative integer m representing the number of points scored by a basketball team, calculates the number of distinct ways in which that score could have been accumulated. Or, to put it more simply (and without mentioning basketball), have the program calculate the number of distinct sequences whose elements sum to m and each of whose members is either 1, 2, or 3.

Hint: There is exactly one way to score zero points (corresponding to the empty sequence), and there are no ways to score fewer than zero points. To score m points, for $m > 0$, one can score $m - 3$ points (in any way possible) followed by a 3-point field goal, or score $m - 2$ points (in any way possible) followed by a normal field goal, or score $m - 1$ points (in any way possible) followed by a free throw.

Input: The first line contains a positive integer n indicating how many instances of the problem are to be solved. Each of the following n lines contains a single nonnegative integer m representing a number of points.

Output: For each number m provided as input, display it, followed by the word `points` and a colon, followed by the number of distinct ways a basketball team could accumulate m points, as described above, followed by the word `ways`.

Sample input	Resultant output
4	0 points: 1 ways
0	3 points: 4 ways
3	10 points: 274 ways
10	13 points: 1705 ways
13	