University of Scranton
ACM Student Chapter / Computing Sciences Department
27th Annual High School Programming Contest (2017)

Problem 1: Divisor Differences

Develop a program that, given integers m and k satisfying $m > k > 0$, lists every pair of positive integers (i, j) such that $j - i = k$ and both i and j are divisors of m.

Input: The first line contains a positive integer n indicating how many instances of the problem are described on the succeeding n lines. Each instance of the problem is described on a single line containing two positive integers, m and k, as described in the preceding paragraph.

Output: For each given instance (m, k) of the problem, echo m and k followed by a colon and a space, followed by the list of all (i, j) pairs that are in relation to m and k as described above. The pairs should appear so that the i values are in increasing order.

```
Sample input:          Resultant output:
------------           ----------------
11
12 1                   12,1: (1,2) (2,3) (3,4)
12 2                   12,2: (1,3) (2,4) (4,6)
12 3                   12,3: (1,4) (3,6)
12 4                   12,4: (2,6)
12 5                   12,5: (1,6)
12 6                   12,6: (6,12)
12 7                   12,7:
12 11                  12,11: (1,12)
120 5                  120,5: (1,6) (3,8) (5,10) (10,15) (15,20)
19 2                   19,2:
36 6                   36,6: (3,9) (6,12) (12,18)
```

Problem 2: Diamond/Hourglass Figure

Develop a program that "draws" a figure as illustrated below, of any specified size. The figure depicts two shapes enclosed in a box, one shape resembling a diamond and the other resembling an hour glass. Exactly what is meant by "size" will be evident from viewing the sample figures below.

Input: The first line contains a positive integer n indicating how many instances of the figure are to be drawn. Each subsequent line contains a positive integer indicating the desired size of the figure to be drawn.

Output: For each size provided as input, a figure of that size is to appear, followed by a blank line.

Sample input and resultant output appear on the next page...

Sample input:

```
3
5
4
1
```

Resultant output:

```
+---------+
|    *    |
|   /*\   |
|  //*\\  |
| ///*\\\ |
|////*\\\\|
|\\\\*////|
| \\\*/// |
|  \\*//  |
|   \*/   |
|    *    |
+---------+
|\\\\*////|
| \\\*/// |
|  \\*//  |
|   \*/   |
|    *    |
|    *    |
|   /*\   |
|  //*\\  |
| ///*\\\ |
|////*\\\\|
+---------+

+-------+
|   *   |
|  /*\  |
| //*\\ |
|///*\\\|
|\\\*///|
| \\*// |
|  \*/  |
|   *   |
+-------+
|\\\*///|
| \\*// |
|  \*/  |
|   *   |
|   *   |
|  /*\  |
| //*\\ |
|///*\\\|
+-------+

+-+
|*|
|*|
+-+
|*|
|*|
+-+
```

Problem 3: Run-length Encoding

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:

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

Using parentheses to partition S into its runs, we get

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

The rules for performing run-length encoding 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 shown 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, given a sequence of nonnegative integers, produces its run-length encoding. For example, given S (as shown above) as input, it would produce T (as shown above) as output.

Input

The first line contains a positive integer n indicating how many sequences are to be run-length encoded. 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.

Output

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

```
Sample Input:
-------------
2
13
4 4 4 4 4 4 0 71 71 5 5 5 5
20
1 0 0 1 1 1 1 1 1 1 1 1 2 3 4 0 0 0 2



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

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

Problem 4: Intersection of Two Lines

Recall that, in Euclidean geometry, any two non-parallel lines intersect at exactly one point. Develop a program that, given two such lines, identifies their point of intersection. Each line is described by two points that lie on it.

Hint: Given the coordinates of two points (having distinct x-coordinate values), it is not difficult to compute m and b such that the line passing through them is characterized by the equation $y = mx + b$ (called the *slope-intercept* form). Given two such equations $y = m_i x + b_i$ $(i = 1, 2)$, it is not difficult to compute the unique pair (x_0, y_0) that satifies both.

Input: So as not to confuse the notion of a line of input data with the notion of a line on the cartesian plane, here we shall refer to the former as an "input record".

The first input record contains a positive integer n indicating how many pairs of lines are subsequently described. Each subsequent input record describes a pair of (non-parallel) lines. Each line is described by two points that lie on it. Each point is described by its x- and y-coordinates, respectively. Hence, each input record (after the first) contains eight real numbers, $x_1, y_1, x_2, y_2, \ldots, x_4, y_4$, describing the four points $p_i = (x_i, y_i)$, $i = 1..4$. It is to be understood that the two lines thereby described are $L_1 = \overleftrightarrow{p_1 p_2}$ and $L_2 = \overleftrightarrow{p_3 p_4}$.

Output: For each pair of lines given as input, the point at which the two intersect is to be displayed in the usual (x, y) format. (See sample output below.)

```
Sample input:
------------
5
2.0 5.0 5.0 1.0 -1.0 -5.0 -1.0 4.0
-2.0 1.0 4.0 1.0 6.4 8.7 -0.6 3.0
7.4 -3.8 -1.0 5.5 0.0 4.0 1.4 0.0
7.4 -3.8 8.0 5.5 1.0 0.0 -1.4 0.0
10.0 1.0 0.0 0.0 0.0 -1.0 10.0 0.1


Resultant output:
-----------------
(-1.0,9.0)
(-3.05614,1.0)
(-0.22449,4.641399)
(7.645161,0.0)
(100.0,10.0)
```

Problem 5: Polynomial Derivative

The central notion in *Differential Calculus*, which is concerned with rates of change (e.g., of one quantity in relation to another), is the **derivative**.

Suppose that we have a function f mapping real numbers to real numbers, such as is illustrated below in the usual way (i.e., by a curve drawn on the cartesian plane).

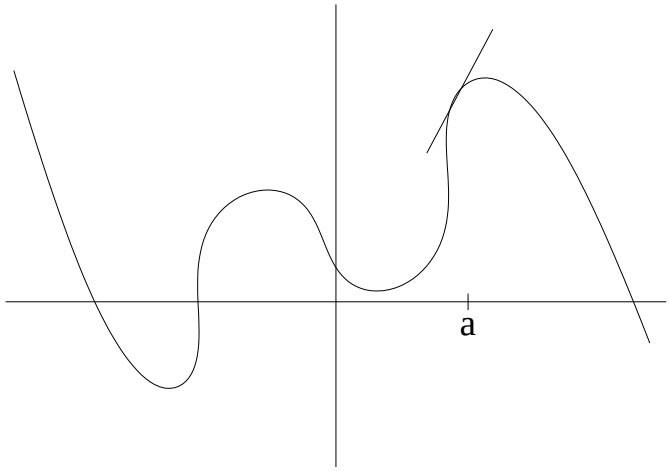

Figure 1: **Curve Depicting a Function** f

Let a be a real number. Suppose that, as in the figure, we draw a tangent line to the curve at $x = a$ (i.e., a line that passes through the point $(a, f(a))$ and has the same slope as does the curve at that point). Then the derivative of f at a is defined to be that slope.

This gives rise to the notion of the *derivative function* of f (usually abbreviated to "derivative of f" and denoted by f'), defined by

$$f'(x) \;=\; \text{the derivative of } f \text{ at } x$$

In other words, for all x, $f'(x)$ is the slope of the line tangent to f at the point $(x, f(x))$.

Remarkably, in many cases, given an algebraic description of a function (that is, an expression involving the variable x, numeric constants, applications of addition, multiplication, etc., and perhaps well-behaved functions from trigonometry (e.g., sin)), one can calculate an algebraic description of its derivative using simple transformation rules.

Among the kinds of functions for which calculating the derivative is easy are those described by *polynomials*. A polynomial is an expression of the form

$$c_0 \ + \ c_1 x \ + \ c_2 x^2 \ + \ c_3 x^3 \ + \ \cdots \ + c_k x^k$$

in which each of the c_i's (referred to as the *coefficients*) is a real constant. That is, it is a sum of terms, each of which is a constant multiplied by x raised to a nonnegative integer power. The largest power (in the example, k), is referred to as the *degree* of the polynomial. Such an expression describes a function of x, of course. It turns out that the derivative of this function is described by the polynomial

$$c_1 \ + \ 2c_2 x \ + \ 3c_3 x^2 \ + \ 4c_4 x^3 \ + \ \cdots \ + kc_k x^{k-1}$$

That is, each term $c_r x^r$ in f becomes $r \cdot c_r x^{r-1}$ in f'. In displaying a polynomial, the standard practice is to avoid the use of unary minus signs and to omit any terms having a zero coefficient (unless there is only one term). For example, the polynomial

$$-2 \ + \ -3x \ + \ 0x^2 \ + \ 13x^4$$

would be written as

$$-2 \ - \ 3x \ + \ 13x^4$$

which avoids using the unary minus in -3 and omits the term $0x^2$. (As -2 is the first term, its unary minus remains.)

Develop a program that, given as input a polynomial with integer coefficients describing a function f, outputs the polynomial describing its derivative, f'.

Input: The first line contains a positive integer n indicating the number of polynomials subsequently given. Each polynomial is described on two lines, the first of which is a nonnegative integer k, $0 \leq k \leq 20$, indicating the polynomial's degree. On the following line appear the $k + 1$ integer coefficients, in the order c_0, c_1, ..., c_k.

Output: For each polynomial given as input, display it on one line, its derivative on the next line, and make the next line blank.

```
Sample input:
-------------
2
5
4 -2 12 0 -1 3
0
14

Resultant Output:
-----------------
4 - 2x + 12x^2 - x^4 + 3x^5
-2 + 24x - 4x^3 + 15x^4

14
0
```

Problem 6: Triangle Path Maximum Sum

Consider the following triangular matrix of numbers.

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

A *path* through the matrix includes exactly one element from each row. For two elements from adjacent rows to be in the same path requires that one element be either directly below the other, or else below it and one place to the right. In other words, in traversing a path from the top row to the bottom, at each step you must move either to the element directly below or to that element's immediate right neighbor.

The *sum of a path* is simply the sum of the elements in that path. In the 7-row matrix depicted above, the sum of the path whose elements are underlined is 37, which is the maximum of the sums of all 64 paths.

Develop a program that, given a triangular matrix, computes the maximum of the sums of all the paths through it.

Input: The first line contains a positive integer n indicating how many triangular matrices are subsequently described. Each matrix is described on $m + 1$ lines, where the first line contains m (with $m > 0$) and each of the next m lines contains the elements of one row of the matrix. For i satisfying $1 \leq i \leq m$, the ith row contains exactly i integers, separated by one or more spaces.

Output: For each matrix given as input, the maximum of the sums of all the paths through it should be displayed on a line in a format consistent with the sample output shown below.

Caution: The input data used for testing your program will include matrices sufficiently large to ensure that a program that traverses *all* its paths (or even a relatively small fraction of them) will require years of execution time before completing. (Note that a matrix of n rows has 2^{n-1} distinct paths through it. If, say, $n = 100$ and a program were to compute the sums of paths at a rate of 1000 paths per second, it would take billions of years before all sums had been computed.) Thus, for you to earn credit for solving this problem will require that you devise a program that employs a much more efficient approach.

Sample input and output appear on the next page ...

```
Sample input:          Resultant output:
------------           ----------------
3                      37
7                      52
5                      191
6 2
0 3 8
4 9 5 1
8 4 0 4 2
1 0 5 7 4 0
9 4 2 5 0 3 7
8
 3
 5 -2
-3  4  8
 7  2 -1  5
 2  9 14 -2 17
10 -3  8  6  2 -2
 8 -2  5  7 13 -9 -5
-5  0  3  6 -2  5  9 13
14
 13
 -4  14
  9   8   3
 16 -13  18  -1
  3   0   4 -19  17
 -7  21   0   5  -8  11
 41  15  32  -3  16   7   5
-12   0   3   5  -7   9   1  -4
  0   8   5  37   2  -1   8  -9  14
  9  17 -21  -1   5  16   3  19   5  15
 25   4  -1   5  12  -6   6   7   9   3   1
  0   5   9 -11   4   2  10  -3   2   0  15   0
  2  -1   8   4  23   3  -1   4  -5  12  -8  13  53
  5   7  -6  20   0  15   2  16 -10   2  29   2  -3   7
```

Problem 7: Prefix to Fully-Parenthesized Infix

Arithmetic expressions are composed of numerals, variables, operators, and parentheses. Here are four examples:

$$x + 5 - y \qquad 35 * (3 + y) - x \qquad x \qquad (7 + z * 4) - y/2$$

All but the third would be ambiguous except for the fact that we assume that the multiplicative operators ($*$ and $/$) have higher precedence than the additive operators ($+$ and $-$) (e.g., so that $7 + z * 4$ means the same thing as $7 + (z * 4)$) and that operators in the same category associate to the left (e.g., so that $x + 5 - y$ means the same thing as $(x + 5) - y$.

Suppose that we had no such rules of precedence or association. Then we would probably resort to using *fully-parenthesized* expressions in which, for every operator, there appears a corresponding pair of parentheses that enclose that operator's scope. The fully-parenthesized equivalents of the expressions above are

$$((x + 5) - y) \qquad ((35 * (3 + y)) - x) \qquad x \qquad ((7 + (z * 4)) - (y/2))$$

By convention, we write arithmetic expressions using *infix* notation, which is to say that operators appear in between their two operands. An alternative notation is called *prefix*, in which each operator precedes its two operands. Interestingly, expressions in prefix notation have no need for parentheses at all, even absent any rules of operator precedence. For example, using prefix notation the expressions above would be written as follows:

$$- + x\ 5\ y \qquad - * 35\ +\ 3\ y\ x \qquad x \qquad - + 7 * z\ 4\ /\ y\ 2$$

Develop a program that, given a prefix arithmetic expression, translates it to the corresponding fully-parenthesized infix expression.

Input: The first line contains a positive integer n indicating how many prefix arithmetic expressions will appear on succeeding lines. Each of the next n lines will contain one such expression. Each item (i.e., operator symbol, numeric literal, or variable name) appearing in an expression will be separated from its neighbor(s) by a space. The only four operators that may appear are those appearing in the examples above. Variables names will be single letters. All numeric literals will be sequences of one or more decimal digits.

Output: For each given prefix expression, output it on one line, its equivalent fully-parenthesized infix expression on the next line, followed by a blank line. There should be a single space on each side of each operator symbol, but no other spaces.

```
Sample input:
-------------
6
x
+ 6 y
+ * 2 3 4
* 2 + 3 4
* + 2 3 4
* + 15 / + z 7 9 / x 2


Resultant output:
-----------------
x
x

+ 6 y
(6 + y)

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

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

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

* + 15 / + z 7 9 / x 2
((15 + ((z + 7) / 9)) * (x / 2))
```

Problem 8: Binary Scientific Notation

Note: This problem is written under the assumption that the reader is familiar with the binary (i.e., base two) numeral system for representing real numbers. See the appendix that follows this problem. **End of note.**

Scientific notation is a convenient way of expressing numbers having very small or very large magnitudes. For example, $.327 \times 10^{12}$ and $.1063 \times 10^{-7}$ are more readable than $327,000,000,000$ and 0.00000001063, respectively. In these two examples, .327 and .1063 are referred to as the **significands** (as they convey the **significant digits**); meanwhile, 12 and -7 are the **exponents** and 10 is the **base**. Both of these examples are in **normalized form**, meaning that the first non-zero digit is in the tenth's column of the significand (i.e., immediately to the right of the **radix point**). The number of digits in a significand is a measure of its **precision**.

The standard scheme by which computers encode real numbers (as opposed to integers) is known as **floating-point** format, which is based on scientific notation. However, computers use the binary alphabet $\{0, 1\}$ (as opposed to the decimal alphabet $\{0, 1, 2, \ldots, 9\}$) to encode numbers (as well as every other kind of data).

Examples of normalized binary scientific notation are $.101101 \times 2^{-3}$ and $.110 \times 2^{5}$, which correspond to .000101101 and 11000, respectively. (Actually, these are in a hybrid form in which the significand is in binary but the exponent is in decimal.) Notice how the exponents -3 and 5 have the effects of moving the radix point three places to the left and five places to the right, respectively, which is analogous to the effect of the exponent in decimal scientific notation.

Develop a program that, given decimal numerals describing a positive real number r and a desired precision p, produces a binary significand $m = .1b_2b_3 \ldots b_p$ (where each b_i is either 0 or 1) and a decimal integer exponent e such that, if $m' \times 2^e$ equals r "exactly", then $m' = .1b_2b_3 \ldots b_p b_{p+1} b_{p+2} \ldots$. (In other words, m is obtained by truncating m' to its first p bits.)

Input: The first line contains a positive integer n indicating how many instances of the problem are described on the succeeding n lines. Each instance of the problem is described on a single line containing a positive real number r and a positive integer p, playing the roles described in the preceding paragraph.

Output: For each given instance (r, p) of the problem, echo r, followed by a colon and a space, followed by a p-bit significand m (with a 1 bit in the 2^{-1} column), followed by an exponent e (in decimal), where m and e satisfy the conditions mentioned above.

Sample input and output are found on the next page ...

```
Sample input:          Resultant output:
------------           ----------------
12
24.875 9               24.875: .110001110 5
24.875 8               24.875: .11000111 5
24.875 7               24.875: .1100011 5
24.875 6               24.875: .110001 5
24.875 3               24.875: .110 5
259.5 12               259.5: .100000011100 9
259.5 5                259.5: .10000 9
0.004 7                0.004: .1000001 -7
0.5 1                  0.5: .1 0
0.085 7                0.085: .1010111 -3
0.085 5                0.085: .10101 -3
0.085 4                0.085: .1010 -3
```

Appendix: Binary Numerals

To remind you how the **decimal** (i.e., base ten) **numeral system** works, take as an example the (decimal) numeral 203.57. The number it represents depends not only upon the values of the digits that appear in it, but also upon their positions relative to the *radix point*. We refer to the positions to the left of the radix point as being the ones', tens', hundreds', etc., columns, corresponding to the nonnegative powers of 10: 10^0, 10^1, 10^2, etc. The positions to its right are the tenths', hundredths', thousandths', etc., columns, corresponding to the negative powers of 10: 10^{-1}, 10^{-2}, 10^{-3}, etc. Thus, considering the digits in right-to-left order,

$$203.57 \;=\; 7 \cdot 10^{-2} \;+\; 5 \cdot 10^{-1} \;+\; 3 \cdot 10^{0} \;+\; 0 \cdot 10^{1} \;+\; 2 \cdot 10^{2}$$

The binary numeral system works the same way, except that only the binary digits (i.e., "bits") 0 and 1 may appear in a numeral and the column "place values" are the powers of two rather than of ten. For example (using subscript 2 to emphasize that this is a binary numeral),

$$1001.011_2 \;=\; 1 \cdot 2^{-3} \;+\; 2 \cdot 2^{-2} \;+\; 0 \cdot 2^{-1} \;+\; 1 \cdot 2^{0} \;+\; 0 \cdot 2^{1} \;+\; 0 \cdot 2^{2} \;+\; 1 \cdot 2^{3}$$

To find the binary numeral representing a positive real number x, we can split x into its integer and fractional parts, find the binary numerals representing each one, and then concatenate them.

In general, the binary numeral representing integer $k > 0$ is zb, where $b = 0$ if k is even and $b = 1$ if k is odd, and z is the binary numeral representing $\lfloor k/2 \rfloor$ (i.e., the integer part of $k/2$). Thus, starting with k, we repeatedly divide by two (until arriving at zero) and concatenate the remainders from right to left. Here we illustrate using $k = 41$:

$41/2 \;=\; 20$, remainder 1
$20/2 \;=\; 10$, remainder 0
$10/2 \;=\; 5$, remainder 0
$5/2 \;=\; 2$, remainder 1
$2/2 \;=\; 1$, remainder 0
$1/2 \;=\; 0$, remainder 1

Placing the remainders from right to left in the order in which they were generated, we get 101001, which is the binary numeral corresponding to (decimal numeral) 41.

Coninuted on next page ...

Now we consider how to find a binary numeral representing a real number y satisfying $0 \leq y < 1$. In general, this numeral is $.bz$, where $b = \lfloor 2y \rfloor$ and $.z$ is the binary numeral that represents $2y - b$. Here we illustrate using $y = .35$:

$\lfloor 2 \cdot 0.35 \rfloor = \lfloor 0.7 \rfloor = 0 = b_1$; $2 \cdot 0.35 - b_1 = 0.7 - 0 = 0.7$
$\lfloor 2 \cdot 0.7 \rfloor = \lfloor 1.4 \rfloor = 1 = b_2$; $2 \cdot 0.7 - b_2 = 1.4 - 1 = 0.4$
$\lfloor 2 \cdot 0.4 \rfloor = \lfloor 0.8 \rfloor = 0 = b_3$; $2 \cdot 0.4 - b_3 = 0.8 - 0 = 0.8$
$\lfloor 2 \cdot 0.8 \rfloor = \lfloor 1.6 \rfloor = 1 = b_4$; $2 \cdot 0.8 - b_4 = 1.6 - 1 = 0.6$
$\lfloor 2 \cdot 0.6 \rfloor = \lfloor 1.2 \rfloor = 1 = b_5$; $2 \cdot 0.6 - b_5 = 1.2 - 1 = 0.2$
$\lfloor 2 \cdot 0.2 \rfloor = \lfloor 0.4 \rfloor = 0 = b_6$; $2 \cdot 0.2 - b_6 = 0.4 - 0 = 0.4$

At this point we notice that we have arrived at $y = 0.4$ for a second time, four steps after the first time, which means that the last four steps (and hence bits b_3 through b_6) will repeat forever! The binary numeral representing .35 is thus $.01\overline{0110}$, the bar indicating that the bit sequence under it repeats forever. This illustrates that a real number that can be represented finitely using a decimal numeral may require a non-finite representation as a binary numeral. Indeed, this is the usual case rather than an exception.

Putting together the binary representations of 41 and 0.35, we get that $41.35 = 101001.01\overline{0110}_2$.