University of Scranton
ACM Student Chapter / Computing Sciences Department
28th Annual High School Programming Contest (2018)

Problem 1: Sums of Groups

Develop a program that, given as input a sequence of integers whose members are to be interpreted as "groups" separated by occurrences of zero, prints the members of each group as well as their sum.

Input: The input data is a sequence of integers on one or more lines. An occurrence of 0 indicates a boundary between two groups. Two 0's in a row signals end-of-input.

Output: For each group, print "`Group:  `" followed by its members, all on one line. On the next line, print "`Sum is `" followed by the sum of the group's members, followed by a blank line.

```
Sample input:
-------------
34 -4 87 12 0 14 2 5 -8 0 4 36
135 -24
0 14 9 0 18 0 0


Resultant output:
-----------------
Group: 34 -4 87 12
Sum is 129

Group: 14 2 5 -8
Sum is 13

Group: 4 36 135 -24
Sum is 151

Group: 14 9
Sum is 23

Group: 18
Sum is 18
```

Problem 2: Colored Balls in Bins

You have a collection of balls, each of which is either red, green, or blue in color. You normally keep them stored in three bins, with all the red balls in one bin, all the green ones in another, and all the blue ones in another. But your younger nieces and nephews came to visit, and when they were finished playing with the balls, they put them back into the bins without regard for color. As a consequence, each of your bins now contains balls of all three colors!

Being someone who suffers from (or is blessed with, depending upon your point of view) OCD, you feel compelled to rectify this situation. Your task, then, is to transfer balls between the three bins to put things right again! The bins are interchangeable, so you don't care which bin contains which balls, as long as each bin contains all the balls of one color.

What you do care about is being efficient, which here means transferring as few balls as possible to achieve your goal. Thus, you have decided that, before you start moving balls around, you are going to figure out, for each color, into which bin the balls of that color should be placed. This calls for a computer program! (Your nieces and nephews are likely to visit many times in the future, so you want to have a program that you can use every time they mess up your bins!)

Develop a program that, given as input a description of how the colored balls are distributed among the three bins, reports, for each color, the bin into which all the balls of that color should be placed, satisfying the condition that the fewest possible number of balls will be transferred from one bin to another.

Input: The first line contains a positive integer n indicating how many instances of the problem are described thereafter. Each instance of the problem is described on a single line containing nine nonnegative integers. The first three indicate the number of red, green, and blue balls, respectively, that occupy the first bin. The next three similarly describe the contents of the second bin, and the last three describe the contents of the third bin.

Output: For each instance of the problem, print a permutation of the letters `R`, `G`, and `B` to indicate which balls should be placed into which bins. For example, `GBR` says that all green balls should end up in the first bin, all blue ones in the the second bin, and all red ones in the third. Following that, print a space followed by the total number of balls that would be transferred from one bin into another to achieve the distribution you describe.

Sample input:	Resultant output:
3	GRB 0
0 1 0 2 0 0 0 0 3	GRB 8
5 6 3 0 0 0 0 0 1	RBG 34
9 12 4 6 3 12 9 13 0	

Problem 3: Draw Staircase

Develop a program that can "draw" a figure as illustrated below, of any specified size. The figure depicts a staircase on each stair of which (except the topmost) stands a (miniature?) person.

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, which corresponds to the number of persons who appear in it.

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

```
Sample input:
-------------
2
4
1

Resultant output:
-----------------
                 o  *******
                /|\ *     *
                / \ *     *
           o  ******     *
          /|\ *          *
          / \ *          *
       o  ******         *
      /|\ *              *
      / \ *              *
  o  ******              *
 /|\ *                   *
 / \ *                   *
**************************

  o  *******
 /|\ *     *
 / \ *     *
************
```

Problem 4: Three Points Determine a Circle

Develop a program that, given as input a triple of (i.e., three) non-colinear points on the cartesian plane, reports the center and radius of the circle on which all three points lie.

Hint 1: The set of points equidistant from points p and q form the perpendicular bisector of the line segment $\overline{pq}$.

Hint 2: Two (non-vertical) lines are perpendicular to each other if (and only if) the product of their slopes is -1.

Input: The first line contains a positive integer n indicating how many triples of points are thereafter identified. Each of the next n lines identifies three non-colinear points, each one described by its x- and y-coordinates, respectively, given by real numbers.

Output: For each given triple of points, the program displays a message identifying those points and the center and radius of the circle on which all of them lie. For the proper output format, see the sample output below. Note that computer arithmetic on real numbers is not exact, so don't be alarmed if your results on the sample data shown below do not match exactly the results shown.

```
Sample input:
------------
2
0.0 0.0 1.0 1.0 -1.0 1.0
0.0 -4.0 1.5 6.4 -2.0 -5.0

Resultant output:
-----------------
The points (0.0,0.0), (1.0,1.0), and (-1.0,1.0)
lie on the circle with center (0.0,1.0) and radius 1.0

The points (0.0,-4.0), (1.5,6.4), and (-2.0,-5.0)
lie on the circle with center (-4.207513,1.915026) and radius 7.25884
```

Problem 5: Longest Increasing Subsequence

If S is a sequence, then any sequence T obtained by removing zero or more (not necessarily contiguous) elements from S is said to be a **subsequence** of S. For example, both $T_1 = \langle 47\ -4\ 5\ 17\ 8\ 11 \rangle$ and $T_2 = \langle 3\ 5\ 8\ 11 \rangle$ are subsequences of $S = \langle 15\ 3\ 47\ 92\ -4\ 5\ 0\ 17\ 8\ 3\ 11 \rangle$.

T_2 has the distinction of being an **increasing sequence**, meaning that each of its elements is larger than the previous one. Thus, T_2 is said to be an **increasing subsequence** of S. Because the length of T_2 (four) is maximum among all of S's increasing subsequences, we also say that T_2 is a **longest increasing subsequence** (LIS) of S. (Notice that we use "a", not "the", because a sequence can have multiple LIS's.)

Develop a program that, given a sequence S of integers, reports the maximum of the lengths of its increasing subsequences.

Input: The first line contains a positive integer n indicating how many sequences are provided as input thereafter. Each sequence is described on two lines, the first of which contains a positive integer giving its length. The second line contains its elements (in order from first to last), separated by spaces.

Output: For each sequence provided as input, the maximum of the lengths of its increasing subsequences is to be printed on its own line.

Note: The test data that will be used in judging your program will include sequences that are sufficiently long so that any solution based upon a brute-force approach (i.e., in which a significant fraction of the 2^n subsequences of a sequence of length n are evaluated) will take too long to complete execution in order to be deemed correct.

```
Sample input:                          Resultant output:
-------------                          -----------------
3                                      4
11                                     6
15 3 47 92 -4 5 0 17 8 3 11            1
6
-3 5 12 13 23 47
8
23 15 13 10 8 5 2 0
```

Problem 6: How Many Ways to Make Change

Suppose that you make a purchase at a store and you are owed one dollar and fifty-eight cents in change. The cashier will probably give you a one-dollar bill and six coins: two quarters, a nickel, and three pennies. This makes sense, because it is the smallest collection of bills and coins whose value is one hundred fifty-eight cents. The largest such collection, of course, consists of one hundred fifty-eight pennies. Obviously, there are many other collections of bills and coins having the same value. But how many? This problem explores that question.

We define the function h as follows: For each nonnegative integer m, $h(m)$ is the number of different collections of dollar bills, quarters, dimes, nickels, and pennies having a value of m cents.

Develop a program that, given a nonnegative integer m satisfying $0 \leq m \leq 8889$, computes $h(m)$. (Any value of m exceeding 8889 is such that $h(m)$ is too large to be stored in a variable of type `int` (or its "equivalent") in most programming languages.)

Note: For large values of m, a "naive" or "brute force" solution to this problem is likely not going to finish computing $h(m)$ within the established time limit. Hence, your program should be written with efficiency in mind. **End of note.**

Input: The first line contains a positive integer n indicating how many instances of the problem are described on the succeeding n lines of input. Each instance of the problem is described on a single line containing one nonnegative integer, m, as described above.

Output: For each given value of m, output a message that reports how many ways there are to make change for m cents, in the format illustrated below.

```
Sample input:          Resultant output:
------------           ----------------
6
12                     12 cents: 4 ways
50                     50 cents: 49 ways
0                      0 cents: 1 ways
2093                   2093 cents: 7281197 ways
202                    202 cents: 1706 ways
8503                   8503 cents: 1800018141 ways
```

Problem 7: Prefix to Canonical Infix

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

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

These are what we call *fully-parenthesized* expressions, because for every operator there is a corresponding pair of parentheses that enclose that operator's scope. By assuming that the multiplicative operators $*$ and / have higher precedence than the additive operators $+$ and $-$ and also that two operators in the same category associate to the left (e.g., so that $a - b + c$ and $((a - b) + c)$ mean the same thing), we can remove some of the parentheses in the above to obtain

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

Indeed, we have kept only those parentheses that were necessary to ensure that the meaning of each expression was unchanged. Expressions such as these, in which there are no unnecessary pairs of parentheses, are said to be *canonical*.

By convention, we write arithmetic expressions using *infix* notation, which is to say that operators appear between their two operands (as in the example expressions above). 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:

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

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

Input: The first line contains a positive integer n indicating how many prefix arithmetic expressions appear in the remainder of the input. Each of the next n lines contains one such expression. Expressions include the four binary arithmetic operators (i.e., $+$, $-$, $*$, and /), one-letter variables (e.g., x), and integer literals (e.g., 37). Occurrences of such elements are separated by spaces.

Output: For each given prefix expression, output it on one line, its equivalent canonical infix expression on the next line, followed by a blank line.

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

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)
```