9th Annual Manitoba Programming Contest
September 26, 2015

General Instructions:

1. This document is printed double-sided. Please read both sides of each page.
2. Submit solutions using the PC^2 software.
3. All input should be read from standard input.
4. All output should be written to standard output.
5. Each problem has a two (2) second time limit for CPU time when executed on the judging data.

UNIVERSITY OF MANITOBA

Sponsors

Google

THE UNIVERSITY OF WINNIPEG

NOTE: TEXT IN RED WAS CORRECTED FROM THE DISTRIBUTED CONTEST PACKAGE.

Problem 1 – Best Kilometre Effort

Every time you go running, you use the GPS function of your phone to track your position over short intervals of time. You can track the distance you've run and your average speed, but since you run different routes on different days, you'd like to know your single best time to run 1 km (=1000 m).

Write a program that takes a list of distances and times, and returns the shortest time required to run 1 km (in seconds). The distances represent straight lines run (in order) along the route, and the times are the time required to run the segment.

The kilometre giving the shortest time can start and end at any point in the run. The distances are given in metres from the previous measurement point. The times are measured in seconds from the previous point. You can assume that the speed between points is constant and that the total distance of each run is at least 1 km.

Input

The input consists of several cases. Each case consists of a single run, and begins with *n,* the number of distance/time measurements. The integer n is greater than zero and less than 1000. The next *n* lines have two integers: the distance (in metres) and the time (in seconds), in that order. Both the distance and time are integers that are greater than zero, and the distance is less than 1000 m.

After the final case, the number zero appears on a line by itself.

Output

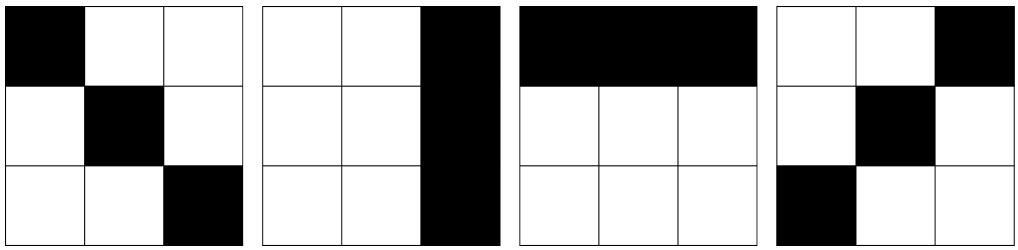
For each run, output the shortest time to run 1 km to the nearest tenth of a second.

Sample Input	Sample Output
4 300 300 500 1001 400 1005 200 207 0	1803.5

Problem 2 – Defective Quilts

Anne has a successful business manufacturing checkered quilts. To maximize profits she uses an automated process to produce them in large volumes. However, she has noticed that some of the quilts contain defects in the form of a series of adjacent black squares (as shown below, either horitzontally, vertically, or diagonally). To solve this problem, she has installed an imaging system that converts an image of each quilt into a 2D matrix, where each value is a number in the interval $[0, 255^3]$. These values represent the colours for each patch of cloth in the image (one for each square), and the colour black is represented as zero.

Anne wants you to develop a program that counts the number (if any) of defects contained in her quilt. To help you with this task, Anne has some additional observations. The defect adjacency length is always odd. If d is the defect adjacency length, there never are any defects in the d/2 (integer division) border elements. Also, patterns of zeros other than those illustrated in the images below are not considered defects, i.e., they are part of her design.



Input

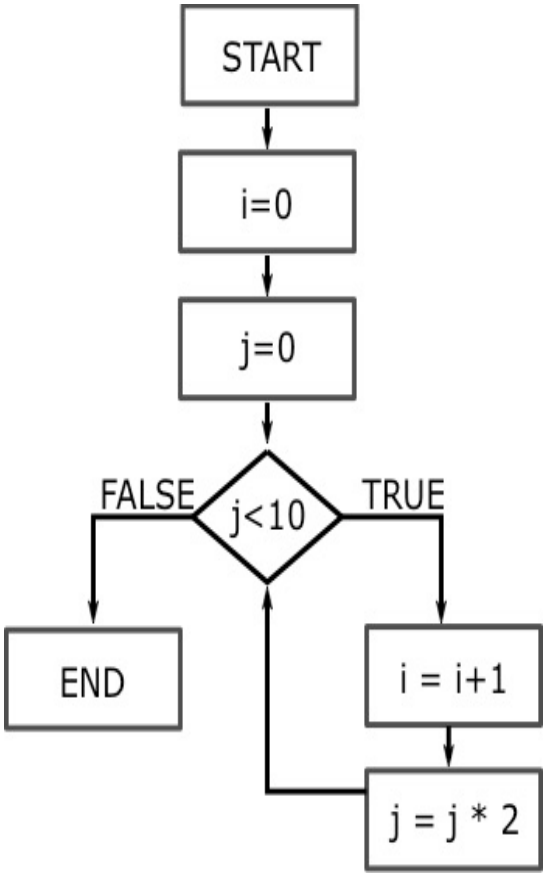
Input will consist of the number of quilts to analyze, followed by a blank line. Then, each sample will contain three pieces of information, beginning with a defect adjacency length (in the interval [3, 11]), the number of rows in the square matrix, and finally the matrix of colour values in 2 dimensions. Each sample will be separated by a blank line. There will never be any defects longer than the input provided, but there may be defects of shorter length.

Output

There should one line of output for each sample, containing the total number of defects in the quilt for the given input length, i.e., the sum of the number of all diagonal, vertical, and horizontal defects in the quilt.

Sample Input	Sample Output
1 5 9 1 2 3 4 5 6 7 8 1 9 1 2 3 4 5 6 7 2 8 9 0 0 0 0 1 2 3 3 4 0 5 6 7 8 9 4 1 2 0 3 4 5 6 7 5 8 9 0 1 2 0 0 0 6 6 7 0 8 9 0 0 3 7 4 5 6 7 8 9 1 2 8 3 4 5 6 7 8 9 1 9	1

Problem 3 – Go with the flowchart



A flowchart is a collection of boxes that are connected by arrows. A flowchart represents the function of an algorithm. Diamond boxes represent decisions (like an if statement) and square boxes represent an action (in this question, an assignment statement). The start and ends of the flowchart are also denoted by boxes labeled "Start" and "End". The boxes are connected by arrows that shows how execution of the algorithm continues after visiting a box. Diamond boxes have two arrows labeled "True" and "False" to represent the action when the condition in the box is true or false. Squares only have one arrow leaving them.

A flow chart can be traced to simulate the execution of an algorithm, by starting at the Start box, then following the arrows. When a square box (action) is encountered, the information in the box is used to update the values of variables. When a diamond (decision) is encountered, the current values of the variables are used to decide which box to go to next.

In our flowcharts, there are only simple assignment statements and comparisons. When a variable is first encountered, that is its definition in the program, that is, there are no variable declaration statements.

Write a program that traces a flowchart and computes the values of all of its variables. You can assume that all programs terminate (there are no infinite loops) and that there are no errors in any box contents (all variables used for assignments and comparisons are defined).

Input

The input is given as a series of lines specifying the boxes in the flowchart. Each flowchart begins with an integer n>1 specifying the number of boxes in the flowchart. The next n lines are the boxes of the flowchart (numbered from 0 to n-1 in order). Each box starts with a number representing it, followed by the type

(START,ACTION, DECISION or END). After the type of box, there is additional information that depends on the type of the box:

- for START, the line contains only the number of the next box to go to.
- for ACTION, the line next has a statement "var = exp" where var is a variable and exp is either (1) a variable, (2) an integer or (3) an expression of the form A+B, A*B or A-B where A and B are either integers or variables. The final integer on the line is the number of the next box.
- for DECISION, the line next has a statement "exp1 OP exp2" where exp1, exp2 are either integers are variables, and OP is either <,> or ==. The final two integers in the line are x y where x is the box that should be visited if the expression on the line is true, while y should be visited if the expression is false.
- for END, there is no more information.

Every symbol, integer or variable name is separated from the adjacent symbols on the same line by exactly one space. Variables will be non-empty sequences of lower case (a-z) letters.

Output

For each flowchart, output the final values of all variables that are defined by executing the flowchart. First output a line that identifies the chart, “CHART X” where X is a number starting at 1 for the first chart, and increasing for each chart. Then output a line “a = N” for each variable a defined in the chart. The variables should be output in alphabetic order. There should be a space before and after the “=” symbol for each variable.

Sample Input	Sample Output
7 0 START 1 1 ACTION i = 0 2 2 ACTION j = 1 3 3 DECISION i < 10 4 6 4 ACTION i = i + 1 5 5 ACTION j = j * 2 3 6 END 0	CHART 1 i = 10 j = 1024

Problem 4 – “eleven points off the word quagmire”

Boggle is a game where players search a grid of letters for words. Words are formed by starting at any letter in the grid and moving in any direction (vertical, horizontal or diagonal) to another letter. For example in the following grid, the word “COMPUTER” appears starting in the first row and third column, and proceeding through the letters highlighted in bold. In Boggle, a letter can only be used once to spell a word.

B	D	**C**	F
M	I	**O**	**M**
N	K	**E**	**P**
G	**R**	**T**	**U**

Write a program that accepts a grid of letters and bounds *n* and *m* and *k* and prints the first and last *k* sequences (in lexicographical order) of letters whose length is between *n* and *m*, inclusive. The printed sequences do not need to be English words – they only need to be sequences of letters that appear in the grid.

Lexicographical order puts words in alphabetical order. If one word is a prefix of the other, then the shorter word comes first. For instance, the word DOG comes before DOGS in lexicographical order, which in turn comes before the word DOT.

Input

Each test case starts with four numbers: r, n, m and k, on a single line. The number r is the size of the grid, n and m are the bounds on the word lengths, and k is the number of words to print from the beginning and end of the list. On the next r lines, there are r upper-case letters (A-Z) on each line. There are no spaces in the grid.

The bounds on the integers are $1 \leq r \leq 8$, $1 \leq n,m, k \leq 10$.

Output

For each test case, first output “Game #x” on a line by itself, where x is a number starting from 1 for the first case. The next two lines out output should each contain k words. The first line should list the first k words (in lexicographical order) of the grid, while the second line should list the last k words (in lexicographical order).

Sample Input	Sample Output
4 3 4 10 BDCF MIOM NKEP GRTU 5 2 2 2 KLMNO ABCDE PQRST UVWXY FGHIJ 0	Game #1 BDC BDCF BDCI BDCM BDCO BDI BDIC BDIE BDIK BDIM UTKR UTP UTPE UTPM UTPO UTR UTRE UTRG UTRK UTRN Game #2 AB AK YT YX

Problem 5 – Butcher

Dale is a butcher, and to streamline his business he produces kubasa (a type of sausage) in extremely long lengths. Dale's resale customers want their meat in different lengths depending on their specific product. Produce a program that takes customer order lengths and produces the point at which Dale's machines should cut the meat. However, there is one catch. Sometimes Dale's machine needs the starting location of each cut, and other times its needs the ending point. To minimize confusion for the customers, the kubasa should be provided in the same order as the customer provided it in the order.

Input

Input will consist of the number of orders for Dale, followed by a blank line. Then, each set of orders will contain three pieces of information, beginning with a binary value (1 representing that the program should output ending point cuts, and 0 represents starting point cut), the number of orders in the list, and finally a single row containing all the orders. Each set of orders is separated by a blank line.

Output

The cut points for each order should be given on a single line separated by spaces (one space between each integer on a line). Each new set of cut points should be on a new line.

Sample Input	Sample Output
2 1 8 3 1 7 0 4 1 6 3 0 6 4 9 1 3 4 9	3 4 11 11 15 16 22 25 0 4 13 14 17 21

Problem 6 – Last Ten and Streaks

In sports league play, two common pieces of information given to evaluate a team's recent performance are the L10 (the "last ten" -- how many wins and losses the team has in its last ten games played) and the "streak" (the maximum number of either wins or losses the team has in a row in its most recent games). For instance, if a team has an L10 of 5-5 and a streak of 3W, then it has won its last three (but not last four) games, as well as five of its last ten games (including the last three).

Given these two pieces of information, how many possible outcomes of the last ten games can there be? If a team has L10 of 1-9 and a streak of 9L, then the sequence of games must have been WLLLLLLLLL (where L is loss and W is win). However, if the team has a L10 of 8-2 and a streak of 7W, then there are two possibilities for the last ten games: LWLWWWWWWW and WLLWWWWWWW.

Input

Each line represents the L10 and streak information for one team. The first two numbers represent the wins and losses (in that order) for the L10 of the team. The two numbers are both non-negative and they sum to ten. The next number is the number for the streak, which is a positive number. Finally, there is a space between the streak number and the type of streak, represented by either W or L.

Output

Each line of output should start with "Team #x:" where x is a number representing the number of the team, starting at 1. For each L10 and streak information, output the number of possible combinations of the last ten games that could produce those L10 and streak statistics.

Sample Input	Sample Output
1 9 9 L 8 2 7 W 5 5 1 W	Team #1: 1 Team #2: 2 Team #3: 70