Where on Earth am I?

You and some friends are on a back country camping trip when your GPS unit fails. It turns out that the onboard software was corrupted by some bizarre atmospheric electrical surges from the storm last night. Your electrical engineer friend was able to hook the GPS to your trusty Palm Pilot, but the algorithms still need to be written to get you back safely to your car. Being a seasoned software veteran you assure your friends there is nothing to worry about and you'll be back to civilization in no time. Of course you will need to test your software before trusting your lives on it.

To determine your location and altitude, three sets of satellite data will be required. Each set of data will come to you as x, y, z and d - where d is the distance between you and the satellite and x, y and z are read as Cartesian coordinates.

Due to the curvature of the Earth the satellites will be less than 200 units from your coordinates and your highest altitude will be 10 units. Your camp will never have negative units of altitude.

Use 10e-6 as your tolerance value for any approximations and there will be a solution for every case. Coordinates will always be whole numbers.

Disclaimer :: The specifications given in this problem do not accurately reflect the way the Global Positioning Satellite system works.

Input Format

The input, located in file prog1.in, will consist of one or more data sets. Each data set will have three lines consisting of the X, Y, and Z coordinates followed by the distance. Each of the coordinates and the distance will be a real number separated by white space.

The input will end with end of file.

Output Format

The output should print out data for each data set. The output for each data set takes the following format:

> The first line contains the test case number (starting with 1).
>
> The next line contains the location where you are located. Each data set will be followed by a blank line.

After the output for all data sets, "End of Output" should be printed.

Sample Data

Example Input

```
20.0 40.0 101.0 106.0660172
40.0 20.0 120.0 124.181319
50.0 50.0 113.0 130.2843045
-10.0 0.0 105.0 119.7038011
0.0 -25.0 100.0 103.4842983
10.0 -31.0 114.0 114.0043859
```
<eof>

Example Output

Test Case #1
-5.0, 0.0, 6.0

Test Case #2
20.0, -55.0, 3.0

End of Output

Trivial Alta-Vista

Your company has an extensive web site and you have been tasked with creating a simple search engine for it. Your program will read one or more keywords and the text to be searched (in html format). It will output all lines containing the keyword(s) found in the text. Only document text should be included in the search results, i.e. html tags and words within tag boundaries will not be 'hits' with this search engine. The tag <META KEYWORD="text"> is an example of what will not be 'found' with your program. HTML tags will begin with a less-than sign < and end with a greater-than sign >. Searches will not be case sensitive. The maximum length of a line of text will be 80 characters.

In the interest of keeping our program trivial, only the following punctuation will be acceptable for producing 'hits' (white space will always produce a hit). They must immediately precede or follow the text being tested.

1) periods
2) commas
3) semicolons
4) colons
5) question marks
6) exclamation points
7) parentheses
8) end of line

Input format

The input file, named prog2.in, will contain one or more data sets. Each data set will contain the following:

The first line of the input will contain the number of keywords to be entered. (N < 10)

The next N lines will contain a keyword.

The next line will contain a blank line.

The next lines will contain the HTML to search. The data set will end with either a blank line or end of file.

Output Format

The output will consist of output data for each data set. The output for each data set will take the following format:

The first line will contain the search number (count from 1)

The next line will contain the keyword number (count from 1) and the keyword.

The next lines will contain the line number of each line which matched the keyword, followed by a dash and the line of text. If the keyword is not found in the text, print "No matches found".

Place a blank line after the last line printed for each keyword.

After the last data set is completed, "End of Output" should be printed.

Sample Data

Input

3
html
web
is

<html>
<head>
<title>Simple Web Page</title>
</head>
<body>
My first link
</body>
</html>

1
is

<html>
<head>
<title>Quotes :: Press On</title>
</head>
<body>
Press on -- nothing in the world can take the place of persistence.
Talent will not; nothing is more common than unsuccessful men with talent.
Genius will not; unrewarded genius is almost a proverb.
Education will not; the world is full of educated derelicts.
Persistence and dedication alone are omnipotent.
-- Calvin Coolidge
</body>
</html>
<eof>

Sample Output

Search #1
Keyword #1 – html
No matches found.

Keyword #2 – web
3 - <title>Simple Web Page</title>

Keyword #3 – is
No matches found

Search #2
Keyword #1 – is
7 - Talent will not; nothing is more common than unsuccessful men with talent.
8 - Genius will not; unrewarded genius is almost a proverb.
9 - Education will not; the world is full of educated derelicts.

End of Output

Down with Rabbitears

Earl E Adopter is tired of his local cable company, Slow Telegouging, Inc. He is considering going with one of the upstart direct satellite system (DSS) companies that provide 18-inch dishes and service. Unfortunately, there are a lot of different options with different costs, so he has decided to write a program that calculates what his best choice is. The program must determine which provider offers the cheapest package that meets Earl's needs. Earl may be moving in the future, so the program must take into account the time over which he is willing to amortize (divide) the costs.

Problem description:

Determine what option has the lowest monthly cost and meets the channel requirements.

Monthly price consists of the monthly fee plus the amortized cost.

Earl is willing to amortize the initial purchase cost over a certain number of months. The DSS company also has a minimum contract requirement. If the DSS minimum contract time is shorter than Earl's time limit, Earl will amortize the initial purchase cost over his time. If the DSS contract requirement is longer than Earl's time, then in addition to the initial cost, Earl will amortize the cost of the extra months required over his time period, in case he moves and can't use the DSS service for the length of the contract.

If multiple packages are tied for the same price, Earl will choose the one with the most channels, or the first package listed if multiple packages offer the same number of channels.

Input format:

All channels are 2-4 character alphanumeric {A-Z, a-z, 0-9} identifiers.

All prices are positive, real numbers; no dollar sign is used.

Provider and package names are words up to 32 characters long, without whitespace.

There will be no more than 10 providers and no more than 10 packages per provider.

No line is longer than 200 characters.

The input filename will be "prog3.in" and may contain multiple datasets.

The first line of the dataset lists the number of months Earl is willing to amortize the cost over (integer), followed by the channels he needs, separated by single spaces.

The second line of the dataset lists the number of service providers.

The remainder of the dataset gives the costs of each provider and the packages that provider offers. Within the section for each provider, the first line lists the name of the provider, the initial purchase cost, the number of months the contract must be kept (an integer), and the number of plans the provider offers. The packages are then listed, one per line. Each line consists of a package name (no whitespace), monthly cost, and a list of channels.

The file ends immediately following the last line of the last dataset. (i.e. Input terminates with End of File.)

Output format:

Each dataset will be preceded by the line "Provider set #:" where # is the dataset's order in the file, counting from 1. The output for each dataset shall consist of a list of all providers and packages, one package per line. All packages meeting the channel requirements will be listed first, followed by the packages that do not meet the requirements. Within each group, the packages will be sorted from lowest to highest cost. Each line lists the provider, package name, and cost rounded to the nearest penny. Packages that do not meet the requirements will have an asterisk at the end of the line. The output for each dataset will be followed by a blank line.

Output will be directed to standard out.

Sample Data

Sample input:

```
12 ESPN CNN TNT MTV HBO ACM
2
Uno 100.00 12 3
Basic 12.00 ESPN CNN TNT MTV
Expanded 18.00 ESPN CNN TNT MTV VH1 CACC
Bonus 25.25 ESPN CNN TNT MTV VH1 CACC HBO ACM
Dos 0 24 2
Silver 25.00 USA GRDN TLC HBO
Gold 30.00 USA TNT ESPN GRDN ACM TLC HBO CACC MTV CNN
```

Sample Output:

```
Provider set 1:
Uno Bonus 33.58
Dos Gold 60.00
Uno Basic 20.33 *
Uno Expanded 26.33 *
Dos Silver 50.00 *
```

I Think I Need Bifocals

When Usenet first began, netizens took pride in the quality of their ASCII art. As time has gone on, the rise of the “luser” and B-grade monster browsers has brought a decline in the quality of art netizens can expect in text-based messages (when people bother to use such a primitive format). One of the problems is size; some of the ASCII images are too small and other images are much too large, sometimes several screens wide. As head programmer at Usenet Central, you’ve decided to fix this problem with a program to improve some of the ASCII images. The program must magnify images that are too small and reduce images that are too large.

Problem Description:

You are given an ASCII art picture as input, and the magnification it must be produced at. Magnifications can be positive or negative (reducing). The pictures consist of ‘X’, ‘/’, ‘\’, and spaces. Each of these characters can be magnified perfectly, as follows for double and triple magnification:

```
Normal      2           3
X           \/          \ /
            /\           X
                        / \

/            /            /
            /            /
                        /

\           \           \
             \           \
                          \
```

A positive magnification, such as 5, means that one character will become a 5x5 character. A negative number, such as –3, means that each 3x3 block will become one character. It will be possible to reduce all images by the given factor.

Input Format:

The input file is named “prog4.in” and will contain one or more data sets. Each data set will have the following format:

The first line consists of three integers, the width, height, and magnification, in that order. The picture follows, with each row of the picture starting at the beginning of a new line. The picture consists entirely of the characters ‘X’, ‘/’. ‘\’, and spaces. Other than the newlines, all characters in the file are in the picture. The maximum width and height are 500. The resulting image will not be more than 500x500.

The input will terminate with end-of-file after the last image.

Output Format:

The output, sent to standard out, consists of the magnified pictures. Each picture is preceded by a line listing “picture #:” where # is the dataset in the file, counting from 1. The picture will start on a new line. A blank line follows each picture.

Sample Data

Sample Input:

```
2 2 2
\X
 X
2 1 2
 X
4 4 –2
\ \/
 \/\
  \/
  /\
```

Sample Output:

```
picture 1:
\ \/
 \/\
  \/
  /\

picture 2:
  \/
  /\

picture 3:
\X
 X
```

Ifs and Fors and Whiles, Oh My!

You’ve just been hired by a major software company that develops some very complicated code. However, they don’t know how complicated their code actually is, and that’s why you’ve been hired. Your job is to implement their Algorithm for Complexity Measurement, which takes in one of their source files, analyzes it, and produces a complexity number for each function in the file.

The way this algorithm works is by rating each statement independently, and then uses the maximum complexity rating within the function as the rating for the function. A regular statement has a rating of 1; an if-then statement is 2; an if-then-else statement is 3; a for loop is 3; and a do-while loop is 5. The rating of an individual statement is the product of the ratings of all the information outside of it. For instance, in the following piece of code:

```
function main() returns INT
declare
   INT i, INT j;
start
   for i = 1 to 10
   start
      j = 1;
      do while (j < 10)
      start
         if (i < j) then
            print(“%s ”, j);        { statement #1 }
         else
            print(“%s “, i);
         j = j + 1;
      end
      print(“\n”);                  { statement #2 }
      if (i == 5) then start
         printf(“\n”);
         printf(“Second half\n”);   { statement #3 }
      end
   end
   return 0;
end
```

statement #1 has a rating of 45 (1*3*5*3), statement #2 has a rating of 3 (1*3), and statement #3 has a rating of 6 (1*2*3), and main has a rating of 45, because that’s the maximum rating of any statement within the function.

Here’s the specs for the language:

First, note comments can be inserted anywhere in the code, with the following exceptions:

- Not within a <string> or <char>
- Not in the middle of a token, which is any set of alphanumeric characters that are contiguous (no whitespace or symbol among them).

<comment>: **{**<any characters excluding the right curly brace>**}**

<program>: <varblock> <function> [<function> […]]
<function>: **function** <ident> **(**[<type> <ident> [**,** <type> <ident> […]]]**)**
  returns <type> <varblock> <block>
<varblock>: **declare** <type> <ident> [**,** <type> <ident> […]]**;**
<block>: **start** <statement>**;** [<statement>**;** […]] **end** | <statement>**;**
<type>: **INT** | **REAL** | **CHAR** | **STRING**
<statement>: **if (**<expression>**) then** <block> [**else** <block>] |
  do <block> **while (**<expression>**);** |
  do while (<expression>**)** <block> |
  for <ident> **=** <integer> **to** <integer> <block> |
  return <expression>;
  <expression>**;**
<expression>: <expression> <operator> <value> | <value> | <unary> <expression> | <function call>
<value>: **(**<expression>**)** | <ident> | <constant>
<constant>: <number> | <string> | <char>
<string>: **“**[<stringchar>[…]]**”**
<char>: **‘**<stringchar>**’**
<number>: <integer>[**.**<integer>]
<integer>: <digit>[<digit>[…]]
<digit>: {**0123456789**}
<operator>: **+** | **-** | **/** | **%** | **>**|**<**|**>=**|**<=**|**=**|**or** | **and**
<unary>: **-** | **not**
<function call>: <ident> **(**[<ident [**,** <ident> […]]]**)**

An <ident> is defined as one or more of the following characters: upper-case alphabetic, lower-case alphabetic, or the underscore. This excludes any keywords (e.g. **for**, **return**, **and**)
A <stringchar> consists of zero or more of any combination of regular characters and special characters. A regular character is any displayable character (e.g. uppercase, lowercase, space, symbols) excluding the backslash (****), the double-quote (**“**), and the single-quote (**‘**). A special character is typed as the backslash followed by another character, but only represents one character. These are used to represent characters that are difficult to display; they are listed in the table below:

Text	Character
\n	Newline
\r	Carriage Return
\”	Double-quote
\’	Single-quote
\b	Backspace
\t	Tab
\\	Backslash
\0	Null character

Input Format:

Input will be read from **prog5.in**. The input consists of a full, syntactically correct, source file. Input terminates with End of File.

Output Format:

The first line of the output is "N **`function(s) in source code`**", where N is the number of functions from the input.

The rest of the output consists of N lines, and each line lists the name of the function, followed by a colon and one space, and then the complexity rating for that function.

Sample Data

Input Sample #1:

```
function main() returns INT
declare
   INT i, INT j;
start
   for i = 1 to 10
   start
      j = 1;
      do while (j < 10)
      start
         if (i < j) then
            print("%s ", j);         { statement #1 }
         else
            print("%s ", i);
         j = j + 1;
      end
      print("\n");                   { statement #2 }
      if (i == 5) then start
         print("\n");
         print("Second half\n");    { statement #3 }
      end
   end
   return 0;
end
```

Output Sample #1:

```
1 function(s) in source code
main: 45
```

Input Sample #2:

```
function safe_sqrt(REAL k) returns REAL start
   if (k >= 0) then
      return sqrt(k);
   return -1;
end

function sq(REAL c) returns REAL
start

   return c*c;
end

function main() returns INT
declare
   REAL a, REAL b, REAL c,
   REAL temp;
start
   do while (1) start
      if (read(“%f %f %f”, a, b, c) < 3) then
         return 0;
      temp = sq(b) – 4*a*c;
      if (temp < 0)
      then
         print(“No \’real\’ roots.\n”);
      else
         print(“%lf %lf\n”, (b + safe_sqrt(temp)) / (2*a),
                            (b – safe_sqrt(temp)) / (2*a));
   end
   return –1;
end
```

Sample Output #2:

```
3 function(s) in source code
safe_sqrt: 2
sq: 1
main: 15
```

Ancient Catacomb Maps

Archaeologists have just discovered some scrolls from an ancient civilization at the recently excavated entrance of a catacomb. They've translated the scrolls and concluded that they're some sort of map to the neighboring catacombs. So, you, as the world-renowned expert in parsing and search algorithms, have been called in to figure out what format the map is in, and how to use it to explore the catacombs.

Upon glancing at the scrolls, you instantly deduce that the map is a set of angles in degrees. Considering this civilization was very advanced in mathematics and engineering, they must have built the catacombs in a set of concentric rings. The map describes the location of the doors in the rings, and the location of walls within the corridors between rings. Thus, the archaeological finds of the millenium must lie within the central chamber.

Your conclusion is that you should write a program to take in several translated maps and to determine the best method for getting from the outside to the center of the catacomb. Knowing that this civilization cherishes efficiency, only one path will be free of booby traps. However, because of efficiency, it will be the shortest path (the one through the fewest doors). If there are two or more distinct paths, then at the first door they differ, choose the one that goes through the inner ring instead of the outer ring, or if they both go through the same ring, choose the one that has the lower degree label.

Input Format:

Input will be read from **prog6.in**.

The first line of the input consists of a positive integer listing the number of maps to follow. There will be at least one blank line between each map, and between the number of maps and the start of the first map. Each of the maps has the same format. A map starts with a line containing a positive integer listing the number of rings in this catacomb. The rest of the map consists of alternating lines describing the locations of doors in the ring and the locations of walls in the corridors. These are listed from outermost to innermost. Note that the map always begins and ends with a ring description, since to get a corridor, you must have an inner and outer ring around it.

The lines describing rings list the number of doors in the ring, followed by the location of the doors expressed as angles in degrees from the center of the catacomb, in no particular order. The lines describing the corridors have a similar format, listing the number of walls followed by the location of the walls expressed in degrees, again in no particular order. All the angles are represented as integers between 0 and 359.

You notice that no map has more than 10 rings, no ring has more than 20 doors, and no corridor has more than 20 walls. You also notice that no map has a wall at the same angle as a door of the inner or outer ring surrounding its corridor.

Output Format:

The output consists of the listing of doors, in order, that need to be passed through to get to the center of the catacomb. List each door on a separate line with the ring number (the outermost ring is ring #1), and followed by the door location angle. Remember to use your knowledge of this civilization to determine the best path, if there is more than one. Place a blank line between each catacomb path outputted.

However, knowing that this was a clever people, you should anticipate some central chambers to be built as unreachable, and thus have the program print out “No solution” if there is no path at all.

Sample Data

Sample Input:

```
2

5
2 20 235
3  140 0 310
4 110 345 320  150
4 87 133 15  335
3 199 0 70
4 100 151 217 12
7 110 22 298 340 67 158 230
5 52 270 180 136 359
3 45 128 196

1
0
```

Sample Output:

```
Maze #1:
1 235
2 150
2 320
2 345
3 0
4 230
5 196

Maze #2:
No solution.
```

The Internet Bubble

The DOTCOM organization evaluates internet stocks by looking at the price of their stock at regular intervals. They do not expect any stock to consistently increase in value, but are looking at the prices relative to each other. They are trying to determine the longest subset of values in which one price is higher than the previous price. It is acceptable to ignore all prices between two prices so long as the first price in the comparison is less than the second price in the comparison. Consider the following example:

For the string of numbers: 1 10 2 10 3 10 4 10 5 10 7 6 8
The longest subset is 7, composed of the 1,2,3,4,5,7,8

The organization considers a stock to have reached YAHOOIAN proportions only after it has achieved a subset of 1000 entries.

INPUT

The input file will consist of one or more data set.

Each line of the set will contain the stock price (in integers) at the requested time. The set will terminate with a value less than 0.

Input terminates at END OF FILE (EOF).

OUTPUT

Output must be generated for each data set. The first line of each data set will indicate the stock number (Starting with #1). Following that the following must appear:

> If the stock has reached YAHOOIAN proportions, one line stating:
> "The stock has reached YAHOOIAN proportions."
>
> If the stock has not reached YAHOOIAN proportions, the following lines must be printed (N is the longest subset of numbers):
>
> "The stock has not reached YAHOOIAN proportions."
> "The longest subset of prices is N."
>
> After each set a blank line must appear.

Output terminates with "END OF OUTPUT".

SAMPLE DATA

Sample Input

1
10
2
10
3
10
4
10
5
10
7
6
8
-1
1
2
3
… (4 through 998)
999
9
1000
9
-1
EOF

Sample Output

Stock #1
The stock has not reached YAHOOIAN proportions.
The longest subset of prices is 7.

Stock #2
The stock has reached YAHOOIAN proportions.

END OF OUTPUT

The Incredible Task Scheduler

Linda Leader is responsible for scheduling all tasks necessary for her team to complete a project. Mary Manager has instructed Linda Leader to create a schedule enabling her team complete all development tasks in the minimum amount of time. Each time Linda begins a project, she must justify to Mary the time required to complete the project. At each of these meetings, Mary inevitably juggles the resources to make it possible to complete the project a few days ahead of schedule. After several instances of this, Linda decides to write a program to make sure she will not be out scheduled in the future.

Definitions

Duration: The tie from when the first task is started, until the last task is completed. If the first task starts on day 1, and the last task completes on day 3, this would be 3 days.

Effort: The total amount of time expended by any developer. If one developer works from day 1 to 3, and another from day 1 to 7, the total effort is 10 days (3 + 7).

Task Duration

To make Linda's job even more difficult, not all tasks are created equal. As people gain experience they are able to complete increasingly difficult tasks in less time. Each task has a target experience level, and duration for that level. A person with less experience may complete a task, but that person will require more time to complete the task. A more experienced person may complete the task in less time. At some point, a person with less experience cannot complete a task, even if they are given an infinite amount of time. The following table relates the difference between a persons experience level and the time required to complete a task.

Perceived Level = (Person Level – Task Level)	Change in Development Time from basic duration.
-4 or less	Cannot be done
-3	+50%
-2	+30%
-1	+10%
0	No change
+1	-10%
+2	-30%
+3 or more	-50%

After applying a task experience ratio, the task duration is rounded up to the next whole number. For instance, for a perceived level of +2, the duration would be 0.7 * base task duration. If the base task duration were 3, this would yield an actual duration of 3 (2.1 rounded up), but if the base task duration were 10, this would yield an actual duration of 7.

Assigning Tasks

A person will always complete the first task in the initial list assigned to them prior to beginning their next task.

The end goal is to allow the project to complete in the smallest duration possible. If multiple solutions allow this to occur, the solution requiring the minimum effort should be selected. If multiple solutions still exist, priority should be given to the solution which assigns the earlier task in the initial list to the earlier developer in the initial list.

The total effort will not exceed 10000 days.

Input

The input file will consist of 1 or more data set. Each data set will contain the following information:

The first line of input will detail the number of developers available (D <= 10).
The next D lines detail each developer's name (string of less than 10 characters) and the developer level.
The next line indicates the number of assignments (A <= 30).
The next A lines indicate the level and duration for the task. The format for each of these lines will be (where NNN is the task number, L is the level and E is the effort):

NNN L E

Input terminates with end of file.

Output

The output will contain several sets of output, one output set for each input set. Each set of output should contain the following:

The first line of each set should contain the project number (starting with #1).

The next three lines should contain the following text, where D is development Time, E is Effort and P is the number of developer involved:

Development time: D days
Effort Required: E days
People involved: P

After a blank line, a table should then appear indicating the task number, task start day, task end day and developer responsible. Dates should count from 1. This table should be sorted based on the order in the initial task list. This table should contain the following header:

"Task Number Start Day End Day Developer"

A blank line should appear following each output set.

Output should terminate with "END OF OUTPUT"

Sample Data

Sample Input

Everything after the semicolon is present for clarification and will not appear in a real data file.

```
4           ; developers
John  3   ; John is level 3
Joe 0       ; Joe is level 0
Bill 0      ; Bill is level 0
Bob 7       ; Bob is level 7
2           ; Assignment
47 2 3      ; Task 47, Level 2 3 days
51 4 3      ; Task 51, Level 4 3 days
EOF
```

Sample Output

```
Project #1
Development time:  3 days
Effort Required: 5 days
People involved: 2

Task Number     Start Day       End Day Developer
47              1               3               John
51              1               2               Bob

END OF OUTPUT
```