1990 ACM Scholastic Programming Contest Finals
sponsored by AT&T Computer Systems

Problem "A"
Rare Order

A rare book collector recently discovered a book written in an unfamiliar language that used the same characters as the English language. The book contained a short index, but the ordering of the items in the index was different from what one would expect if the characters were ordered the same way as in the English alphabet. The collector tried to use the index to determine the ordering of characters (i.e., the collating sequence) of the strange alphabet, then gave up with frustration at the tedium of the task. You are to write a program to complete the collector's work. In particular, your program will take a set of strings that has been sorted according to a particular collating sequence and determine what that sequence is.

Input

The input consists of an ordered list of strings of uppercase letters, one string per line. Each string contains at most 20 characters. The end of the list is signalled by a line that is the single character '#'. Not all letters are necessarily used, but the list will imply a complete ordering among those letters that are used. A sample input file is shown below.

```
XWY
ZX
ZXY
ZXW
YWWX
#
```

Output

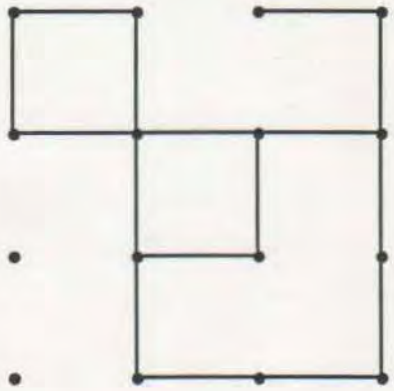
Your output should be a single line containing uppercase letters in the order that specifies the collating sequence used to produce the input data file. Correct output for the input data file above is shown below.

```
XZYW
```

1990 ACM Scholastic Programming Contest Finals
sponsored by AT&T Computer Systems

Problem "B"
Squares

A children's board game consists of a square array of dots that contains lines connecting some of the pairs of adjacent dots. One part of the game requires that the players count the number of squares of certain sizes that are formed by these lines. For example, in the figure shown below, there are 3 squares—2 of size 1 and 1 of size 2. (The "size" of a square is the number of lines segments required to form a side.)



Your problem is to write a program that automates the process of counting all the possible squares.

Input

The input file represents a series of game boards. Each board consists of a description of a square array of n^2 dots (where $2 \le n \le 9$) and some interconnecting horizontal and vertical lines. A record for a single board with n^2 dots and m interconnecting lines is formatted as follows:

Line 1:	`n`	the number of dots in a single row or column of the array
Line 2:	`m`	the number of interconnecting lines

Each of the next m lines are of one of two types:

`H i j` indicates a horizontal line in row i which connects the dot in column j to the one to its right in column $j + 1$

or

`V i j` indicates a vertical line in column i which connects the dot in row j to the one below in row $j + 1$

Information for each line begins in column 1. The end of input is indicated by end–of–file. A sample input file in which the first record represents the board of the square above is on the reverse side of this paper.

problem continued on reverse side →

(Sample input file)

```
4
16
H 1 1
H 1 3
H 2 1
H 2 2
H 2 3
H 3 2
H 4 2
H 4 3
V 1 1
V 2 1
V 2 2
V 2 3
V 3 2
V 4 1
V 4 2
V 4 3
2
3
H 1 1
H 2 1
V 1 2
```


Output

For each record, label the corresponding output with "Problem #1","Problem #2", and so forth. Output for a record consists of the number of squares of each size on the board, from the smallest to the largest. If no squares of any size exist, your program should print an appropriate message indicating so. Separate output for successive input records by a line of asterisks.

Appropriate output for the sample input file is as follows.

```
Problem #1

2 square(s) of size 1
1 square(s) of size 2

**********************************

Problem #2

No completed squares can be found.

**********************************
```

1990 ACM Scholastic Programming Contest Finals
sponsored by AT&T Computer Systems

Problem "C"
Repeating Decimals

The decimal expansion of the fraction 1/33 is $0.0\overline{3}$, where the $\overline{03}$ is used to indicate that the cycle 03 repeats indefinitely with no intervening digits. In fact, the decimal expansion of every rational number (fraction) has a repeating cycle as opposed to decimal expansions of irrational numbers, which have no such repeating cycles. Examples of decimal expansions of rational numbers and their repeating cycles are shown below. Here, we use parentheses to enclose the repeating cycle rather than place a bar over the cycle.

fraction	decimal expansion	repeating cycle	cycle length
1/6	0.1(6)	6	1
5/7	0.(714285)	714285	6
1/250	0.004(0)	0	1
300/31	9.(677419354838709)	677419354838709	15
655/990	0.6(61)	61	2

Write a program that reads numerators and a denominators of fractions and determines their repeating cycles. For the purposes of this problem, define a repeating cycle of a fraction to be the first minimal length string of digits to the right of the decimal that repeats indefinitely with no intervening digits. Thus for example, the repeating cycle of the fraction 1/250 is 0, which begins at position 4 (as opposed to 0 which begins at positions 1 or 2 and as opposed to 00 which begins at positions 1 or 4).

Input

Each line of the input file consists of an integer numerator, which is nonnegative, followed by an integer denominator, which is positive. None of the input integers exceeds 3000. End–of–file indicates the end of input. A sample input file is below.

```
76 25
5 43
1 397
```

Output

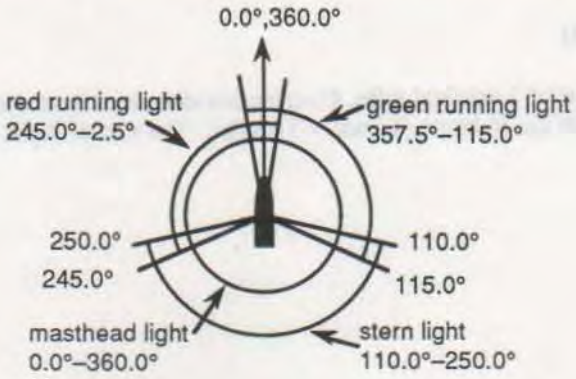
For each line of input, print the fraction, its decimal expansion through the first occurrence of the cycle to the right of the decimal or 50 decimal places (whichever comes first), and the length of the entire repeating cycle. In writing the decimal expansion, enclose the repeating cycle in parentheses when possible. If the entire repeating cycle does not occur within the first 50 places, place a left parenthesis where the cycle begins—it will begin within the first 50 places—and place "...)" after the 50th digit. Output for the sample input file above is shown here.

```
76/25 = 3.04(0)
   1 = number of digits in repeating cycle

5/43 = 0.(116279069767441860465)
   21 = number of digits in repeating cycle

1/397 = 0.(00251889168765743073047858942065491183879093198992...)
   99 = number of digits in repeating cycle
```

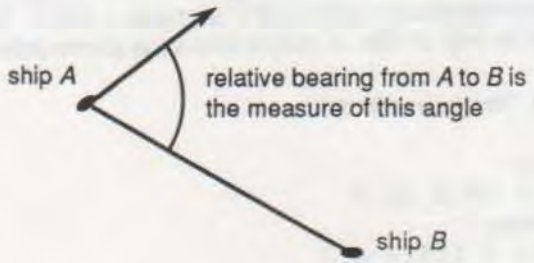
1990 ACM Scholastic Programming Contest Finals
sponsored by AT&T Computer Systems

Problem "D"
Running Lights Visibility Calculator

Ships underway on the high seas at night are required to display navigation lights to identify their location and direction of movement to other ships. Most ships are required to display a set of four running lights—one at the stern (rear), one in the middle on the mast, and two at the bow (front).



Running Lights Configuration



Relative Bearing from *A* to *B*

In naval practice, the course of a ship is the direction the ship is traveling as measured *clockwise* from true north. For example, a ship that is traveling due east is on a 90° course; one traveling on a 315° course is traveling due west–northwest. The relative bearing from ship *A* to ship *B* is the measure of the angle that the course of ship *A* makes with the vector drawn from *A* to *B*, where the initial side of that angle is incident with the course vector and the terminal side is incident with the vector from *A* to *B*. The measurement is taken clockwise.

If we assume that the bow of a ship is pointing to 0.0° or 360.0°, then the running lights have ranges as shown in the figure. Here, the stern (rear) of the ship is at 180.0°. The masthead light shines all directions (0.0° – 360.0°). The stern light shines strictly between 110.0° and 250.0° (the angle at which the stern light is beamed relative to the ship satisfies the inequalities 110.0° < angle < 250.0°). The red running light shines strictly between 245.0° and 2.5°; the green running light shines strictly between 357.5° and 115.0°. (Note the overlap in the visible sectors between the red and green running lights and stern light.) In addition, the nominal maximum light visibility range for all lights is 10 nautical miles (nm).

Write a computer program that will repeatedly read in sets of data describing the location, course and speed of your own ship and other ships in the vicinity. Based on this information, the program will first calculate the relative bearings from other ships to your ship and display the expected configurations of visible lights from left to right as viewed from your own ship. Ships at least 10 nm away will not be visible. The program then recalculates the relative bearings after a 3 minute time delay to determine which ships are on a collision course with your own. If another ship is initially visible and if at the end of the 3 minute delay the relative bearing from that ship to your own remains almost the same (within 2°) while the distance between the ships decreases, then the program must issue a collision warning. Assume that there will be no collisions of any type (ship–to–ship or ship–to–land) in the 3 minute time period.

problem continued on reverse side →

Input

The input file consists of several data scenarios. Each scenario is as follows.

Scenario ID (string)
Number of other ships (integer)
Information on your own ship on two lines:
    name of your ship (string)
    x-coordinate *y*-coordinate course speed (reals)
Other ship information on two lines per ship:
    name of other ship (string)
    x-coordinate *y*-coordinate course speed (reals)

All coordinates are on a cartesian grid with unit measurement of 1 nautical mile. Courses are measured from true north, and each course satisfies $0.0^\circ \le \text{course} < 360.0^\circ$. Speeds are in knots (1 knot = 1 nm/hr). The end of input is indicated by end–of–file. A sample data set is shown below.

```
Sample Test Data Set
4
Ownship
0.0 0.0 90.0 10.0
Windswept
10.0 10.0 135.0 8.0
Footloose
-5.0 6.0 275.0 6.0
Crasher
0.0 9.0 135 14.14
Aquavit
-2.0 -2.0 294.0 15.0
```

Output†

Output consists of a single table per data set. A table shows the ID for each other ship along with its initially calculated relative bearing to your own ship, distance from your own ship, and its light configurations (from left to right) visible from your ship. Specific table spacing is not critical but output should be easily readable. Collision warnings, if any, should appear at the bottom of the table. Each warning should include the name of the other ship and its distance from your own ship at the end of the 3 minute interval. (Do not display the relative bearings, distances, or running lights configurations for the end of that warning interval.) All real output should be written with two digits to the right of the decimal. Separate output for different scenarios with a line of asterisks.

Here is appropriate output for the sample input file above.

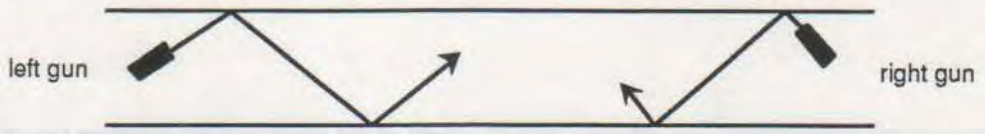
```
Scenario: Sample Test Data Set

Boat ID         Bearing      Distance   Lights (left to right)
-------------------------------------------------------------
Windswept         90.00        14.14     Lights not visible
Footloose        225.19         7.81     Masthead  Stern
Crasher           45.00         9.00     Masthead  Green
Aquavit          111.00         2.83     Stern  Masthead  Green
** Collision warning -->Crasher:  Distance =  8.50
*************************************************************
```

† To help maintain floating point accuracy when converting degrees to radians, use the predeclared constant `pi`.

1990 ACM Scholastic Programming Contest Finals
sponsored by AT&T Computer Systems

Problem "E"
Robot Crash

The DoD company has contracted to determine under what conditions a pair of scanner robots can collide. The robots are fired simultaneously from "guns" that are mounted near opposite ends of a horizontal strip. They travel in straight lines until they hit a wall of the strip or until they are in the same spot at the approximately the same time.



Whenever a robot hits a wall, it bounces off without loss of speed and in a straight line so that the angle of incidence equals the angle of reflection.

If the robots are in the same spot at approximately the same time, then they "collide." Write a program to determine whether robots collide and if so where. To simplify the computer model of the physical problem, assume the following.

1) The horizontal strip is 2-dimensional, and it runs left-to-right. Its walls are straight lines.
2) Each robot is a point mass. That is, the circumference of each robot is 0.
3) A robot maintains the speed with which it was originally fired until it collides with the other robot or until it passes the gun from which the other robot was fired.
4) There are 2 guns, one mounted to the left of the other on a horizontal strip. The initial angle of the left gun is between -85° and 85°. The initial angle of the right gun is between 95° and 180° or –95° and –180°. (All angles are measured counterclockwise from the positive x-axis.)
5) Robots collide when they pass through the same place within 0.5 second of each other.
6) The horizontal strip is 10 units high. For any point (x,y) in the strip, $0 \le y \le 10$.
7) Robots speeds will be positive.

Input

Input for your program is a text file which contains data for several different pairs of robots. The lines of the text file come in pairs. The first line of a pair gives initial firing information about the robot fired from the leftmost gun. The second line of the pair gives initial firing information about the robot fired from the rightmost gun. Each line contains 4 data items as follows:

x-coordinate y-coordinate angle in degrees speed (reals)

The end of input is indicated by end-of-file. Assume that the input is error-free. A sample input file is printed on the reverse side of this paper.

problem continued on reverse side →

(Sample input file)

```
0  4  0  3.3
40  5  125  5
1  6  -5  10
5  2  95  20
2  5  45  5
42 5  -135  5
0 6 20 3
0 5 180 4
```

Output†

For each robot problem, output from your program should consist of the number of the problem (ex: Robot problem #1, Robot problem #2), and a statement indicating whether or not the robots do collide. If they do collide, your program should also print the coordinates of the point of collision. All real output should be printed with 2 digits to the right of the decimal.

Appropriate output for the sample input follows.

```
Robot Problem #1:  Robots do not collide.

Robot Problem #2:  Robots collide at (4.68,5.68)

Robot Problem #3:  Robots collide at (22.00,5.00)

Robot Problem #4:  Robots do not collide.
```

† To help maintain floating point accuracy when converting degrees to radians, use the predeclared constant `pi`.

1990 ACM Scholastic Programming Contest Finals
sponsored by AT&T Computer Systems

Problem "F"
Getting There

A frustrating part of arranging your own air travel trip is selecting from among many possible flights that sequence of flights which will take you from your origin to your destination in the least possible time or for the least possible cost.

It should be clear to any frequent air passenger that in order to reach one city from another, the cost of the shorter flight may be more than the cost of longer flights. In other words, it may pay you well to cool your heels in an airport waiting for a connecting flight rather than take a more direct flight or one in which the connecting time is shorter. For example, consider the following flight schedule.

```
Center City      Homeville        5:20A  6:55A  12.50
Center City      Greenville       5:45A  9:15A  35.00
Homeville        Greenville       7:45A  9:35A  20.00
```

In order to travel from Center City to Homeville, you have two choices. You can travel from Center City to Homeville, then from Homeville to Greenville, or you can travel directly from Center City to Greenville. The first route costs $32.50 and has travel time 4:15; the direct route costs $35.00 and has travel time 3:30. If minimizing cost is your objective, then you would choose the first route. If you want to minimize time, you would select the second route.

You are to write a program to optimize route selection given the criteria of least cost or least time. Your program will read a list of flights and several trip requests and will select from the list of flights the best sequence to satisfy each trip request. For each request, if more than one route should satisfy the request, then your program should select the route that also satisfies the other objective. For example, if cost is to be minimized and if two routes both yield the minimum cost, then select the route which yields the shortest travel time. If two routes yield identical costs and travel times, then select either route.

Input

The input data file is broken into two segments, the first describing the list of flights and the last containing the trip requests. The end of this part of the file is indicated by the line consisting of the single character '#'.

The flight segment of the file describes individual flights, one per line. Each line contains the origin city (columns 1 through 20), the destination city (columns 21 through 40), the departure time (columns 41 through 47), the arrival time (columns 48 through 54), and the cost (columns 55 though 60). City names are left–justified in their respective fields, and may contain upper and lower case characters. Times are in the form *HH:MMX*, where *HH* is the hour (a leading zero may be replaced by a blank), *MM* is the minutes (exactly two digits will appear), and *X* is A (for AM), P (for PM), or M or N (used only after 12:00 to distinguish midnight from noon). The cost of the ticket is in dollars and cents, and includes a decimal point and two fractional digits. No tickets are free or cost more than $999.99. Departure times are always prior to arrival times. No individual flight represented by a line in the schedule takes more than 24 hours. There will be at most 20 flights on the schedule.

The trip request segment of the file immediately follows the list of flights. Each request appears on a line by itself, and specifies the origin city (columns 1 to 20), the destination city (columns 21 to 40), and whether to optimize cost or travel time. If it is desired to optimize travel time, the word TIME is in columns 41 to 44. If cost is to be optimized, then the word COST is in columns 41 to 44. There may be trailing blanks in any line in the flight schedule or the trip requests. The end–of–file indicates the end of the trip requests.

problem continued on reverse side →

Below is a sample input file.

```
Center City          Homeville          5:20A 6:55A   12.50
Center City          Greenville         5:45A 9:15A   35.00
Homeville            Greenville         7:45A 9:35A   20.00
Archer City          Homeville          5:00A 6:00P  612.50
#
Center City          Greenville       COST
Archer City          Greenville       TIME
```

Output

For each travel request, display the request and the optimal route in the following form.

```
From: Center City          To: Greenville              Optimize: Cost
=============================================================================
From                 To                   Leave   Arrive      Cost
=============================================================================
Center City          Homeville            5:20A    6:55A    $12.50
Homeville            Greenville           7:45A    9:35A    $20.00
                                          ----------------------
                                                    4:15    $32.50

From: Archer City          To: Greenville              Optimize: Time
=============================================================================
From                 To                   Leave   Arrive      Cost
=============================================================================
Archer City          Homeville            5:00A    6:00P   $612.50
Homeville            Greenville           7:45A    9:35A    $20.00
                                          ----------------------
                                          1 day 4:35       $632.50
```

The last line gives the travel time in days, hours, and minutes, and the total cost of the trip. All optimum routes will require less than 10 days and less than $1,000.00. Place one blank line between the outputs for successive trips.

1990 ACM Scholastic Programming Contest Finals
sponsored by AT&T Computer Systems

Problem "G"
Meals on Wheels Routing System

The Meals on Wheels program has the responsibility for providing hot meals to homebound senior citizens within the city. Volunteer drivers deliver the meals in a timely manner to ensure that they are still hot on arrival. The list of customers for meals and the number of available drivers varies on a daily basis. For each day, management tries to assign drivers routes so that allocation of customers is as even as possible among the routes.

An algorithm for developing the daily routes involves sorting the addresses of the meal customers based on the directions of their locations relative to the Meals on Wheels headquarters and dividing this sorted list among the available drivers. The Meals on Wheels headquarters is considered to be located at the origin on a cartesian grid of square city blocks. Each customer's address has been converted into the number of city blocks in the x and y directions from the Meals on Wheels facility. For example, a customer living at location (3,–2) would be living 3 blocks east and 2 blocks south of the Meals on Wheels headquarters.

Write a computer program to determine routing for several different days. For each day, your program will read in the number of drivers (routes) and number of customers followed by sets of names and locations for the customers. Allocate customers to routes using the following strategy. Find the polar coordinate of each customer's location. Consider 0° to be due east and 90° to be due north. Sort the sets of polar coordinates by angle and then divide the customers as equally as possible among the available routes starting at the angle of smallest measure. Routes with customers at high degree angles should not have more customers than those for customers of lower degree angles. If two customers are at the same angle, then assign the customer nearer to the Meals on Wheels headquarters before you assign the one further away. The difference in the number of customers assigned to any two routes may not exceed one.

Your program will determine not only the route for each driver but also the total length of each route. The length of any route includes the sum of the distances from the Meals on Wheels headquarters to the first customer, plus the distances between subsequent customers, plus the distance back to the headquarters from the last customer on the route. Note that a block may not be traversed diagonally, and all city blocks are squares.

Input

Input consists of *multiple* data sets in which the first line is a data set ID and the second line contains the number of routes followed by the number of customers. The remaining lines of the data set are arranged in pairs, one pair per customer. The first line of each pair is the customer's name and the second line contains the x and y coordinates of where that customer lives. So each data set is arranged in the following manner.

Line 1: data set ID	(string — maximum length 50 characters)
Line 2: n m	(number of routes number of customers— positive integers)

The next $2m$ lines come in pairs:

Line 3: customer name	(string — maximum length 25)
Line 4: x-coordinate y-coordinate	(x and y coordinates for the preceding customer — integers)

Assume the input is correct and the number of routes does not exceed the number of customers. The end of input is indicated by end–of–file.

A sample input file is printed on the reverse side of this page.

problem continued on reverse side →

(Sample input file.)

```
Sample Route List
4 10
able
1 2
baker
-3 6
charlie
-4 -5
donald
4 -7
eloise
3 4
frank
2 2
gertrude
5 9
horace
-2 -5
inez
5 -3
james
0 1
```

Output

For each data set your output should include the data set ID, the number of customers, the number of routes, the routes of customers in order along with their corresponding route lengths, and the total route length for all routes in this data set. Print a row of asterisks between output for successive data sets. Output for the sample file is below.

```
Sample Route List
Number of Customers:  10          Number of Routes:  4

Route ==>    1
Customer: frank
Customer: eloise
Customer: gertrude
Route Length ==>  28

Route ==>    2
Customer: able
Customer: james
Customer: baker
Route Length ==>  22

Route ==>    3
Customer: charlie
Customer: horace
Route Length ==>  18

Route ==>    4
Customer: donald
Customer: inez
Route Length ==>  24

Total Route Length ==>  92
**********************************
```

1990 ACM Scholastic Programming Contest Finals
sponsored by AT&T Computer Systems

Problem "H"
PGA Tour Prize Money

A PGA (Professional Golf Association) Tour event is a golf tournament in which prize money is awarded to the best players. The tournament is broken into four rounds of 18 holes apiece. All players are eligible to play the first two rounds. Only those with the best scores from those 36 holes "make the first cut" to play the final two rounds and qualify for prize money. Players with the best 72–hole aggregate scores (the lowest scores) earn prize money.

You must write a program to determine how the total prize money (called the tournament "purse") is to be allocated for a tournament. Specifications are as follows.

1) All players will play at least two 18–hole rounds (36 holes in all) unless they are disqualified for some reason.
2) Any player who is disqualified stops playing at the time of the disqualification. Players who are disqualified during the first two rounds are ineligible to make the cut. Players who are disqualified during either of the last two rounds are ineligible to win prize money.
3) At the end of the first two rounds, the field of players is cut to the 70 players with the lowest 36–hole scores plus ties. So if 10 players are tied for 70th place, then 79 players make the 36–hole cut. Players who do not make the 36–hole cut are eliminated from the playing field and do not win any prize money.
4) The players who do make the 36–hole cut play an additional 36 holes (two 18–hole rounds) and are paid a percentage of the total prize money depending on their 72–hole aggregate score. The lower the score, the more prize money a player wins.
5) Players are paid percentages of the the tournament purse according to their final standings. For example, if the tournament purse were $1,000,000 and the winner's share were 18%, the winner would earn $180,000.
6) There will be only one winner of this tournament. (In an actual golf tournament, when there is a tie for the low 72–hole score, there is be a play-off among the tied players. We will ignore that situation.)
7) There may be a tie for any or all of the positions between 2 and 70. If there is a tie among n players for position k, the money designated for positions k through $n + k - 1$ is pooled and allocated equally among the tied players. For example, using the sample data given later, if there were a tie for second place between two golfers, they would each win $88,000 [(10.8% + 6.8%)/2 = 8.8% * $1,000,000]. If there were a three-way tie, all three golfers would get $74,666.66 [(10.8% + 6.8% + 4.8%)/3 = 7.4666% * $1,000,000]. The extra penny is ignored.
8) If disqualification reduces the field to less than 70 players, the money for the last and any other places not covered is not allocated. For example, if exactly 70 players make the cut but three of them are disqualified, then the tournament simply pays 67 places.
9) Amateur golfers may play in professional tournaments but can win no money. Any prize money "won" by an amateur is allocated to the next lower position. For example, if an amateur has placed third in a tournament, then third place money goes to the fourth place finisher, and fourth place money goes to the fifth place finisher, etc.
10) Only the low 70 non–amateur places and ties earn prize money. For example, if 75 players make the 36–hole cut, it is possible for 5 of them not to earn prize money, assuming none of the players making the cut are amateurs.

Input

The input file is broken into two segments. The amount of the tournament purse and the percentages for all the 70 places are stored in the first segment of the input file. This segment contains exactly 71 lines, which are formatted as follows.

Line 1: Total value of the purse
Line 2: Percentage of the purse designated for first place
Line 3: Percentage of the purse designated for second place
...
Line 71: Percentage of the purse designated for 70th place

problem continued on reverse side →

All entries in the first 71 file lines can be read as real numbers. All percentages are given to four decimal places. Assume the percentages are correct and sum to 100%. A partial sample of the first segment of the input file is shown below.

```
1000000.00
18.0000
10.8000
6.8000
4.8000
...
0.2020
0.2000
```

The second segment of the input file contains the players' names and their respective scores for the four rounds. There is a maximum of 144 players. The format of each line is a follows.

Characters 1–20:	Player name
Characters 21–23:	Round 1 score (first 18 holes)
Characters 24–26:	Round 2 score (second 18 holes)
Characters 27–29:	Round 3 score (third 18 holes)
Characters 30–32:	Round 4 score (fourth 18 holes)

Any player who has an asterisk '*' at the end of his last name is an amateur. All players who are not disqualified will have four 18–hole scores listed. (Even though in an actual tournament, players who do not make the cut do not get to play the last two rounds of the tournament, for the purposes of this program all players who are not disqualified will have four 18–hole scores listed.) A player who is disqualified during a round will have a score on that round shown as 'DQ'. That player will have no additional scores for the tournament. Assume that at least 70 players will make the 36–hole cut. The end of input is denoted by end–of–file.

Sample lines from the second segment of the input file are as follows.

```
WALLY WEDGE      70   70   70   70
SANDY LIE        80   DQ
SID SHANKER*    100   99   62   61
JIMMY ABLE       69   73   80   DQ
```

Output

Output from this program consists of names of all players who made the 36–hole cut, their finish positions (with the letter "T" after the numeric value representing the finish position if there is a tie for that position), scores for each round, total scores, and the amounts of money won. Disqualified players are listed at the bottom with scores of DQ placed in the "TOTAL" column; the order among disqualified players is unimportant. No player who failed to make the 36–hole cut is listed in the output. Each column of output should be formatted and labelled appropriately. The dollar amounts should be correct to two decimal places. Sample output is shown below:

```
Player Name          Place     RD1   RD2   RD3   RD4   TOTAL     Money Won
-------------------------------------------------------------------------
WALLY WEDGE          1         70    70    70    70    280       $180000.00
TOMMY TWO IRON       2T        71    72    72    72    287       $ 88000.00
HENRY HACKER         2T        77    70    70    70    287       $ 88000.00
NORMAN NIBLICK*      3         72    72    72    72    288
BEN BIRDIE           3         70    74    72    72    288       $ 48000.00
...
LEE THREE WINES      70        99    99    99    99    396       $  2000.00
EDDIE EAGLE                    71    71                DQ
JIMMY ABLE                     69    73    80          DQ
```