

2000 ACM Mid-Atlantic Regional Programming Contest

Attached you will find the actual packet of questions. A few notes first:

1. There are 8 questions in the packet, numbered 1 – 8. **These questions are NOT necessarily sorted by difficulty.** As a team gets a question correct, they will be brought a balloon. The following table correlates the balloon color to the question:

Question Number	Question Name	Balloon Color
1	Double, Double, Toil and Trouble	Gold
2	Houses Divided	Orange
3	The Computerized Contractor	Silver
4	The "I"s Have It	Blue
5	A Committee is a Beast with Many Heads and No Brain	Purple
6	The New FAA	Green
7	It's time for change	Red
8	Code name: Double-Omega VII	Pink

2. **All questions must read from standard input and write to standard output.**
3. Each submission on a given question will be judged in the order it is received. However, it is possible that if I submit question 2 and then 3, I may get a response from 3 back before 2. Do not submit clarifications asking when a submission will be returned. If a submission has not been judged within 90 minutes, ask the site judge to determine the cause for the delay.
4. Each submission will be judged. The judges will respond with one of the following: If more than one is applicable, the judges may respond with any of the applicable answers.

Response	Description
Correct	The question has been judged correct.
Incorrect Output	The program generated output that is not correct.
Incorrect Output Format	The program generated output that is not in the correct format
Incomplete output	The program failed to generate all required output
Syntax Error	The program failed to compile on the judges machine
Run Time Error	The program experienced a run time error.
Run Time Exceeded	The program failed to complete within the allowable time limit

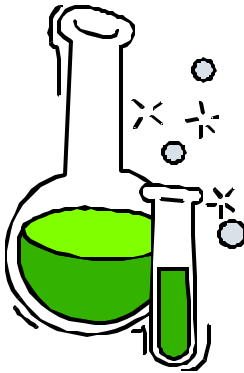
Good Luck and enjoy the contest.

Double, Double, Toil and Trouble

2000 Mid Atlantic Regional ACM Programming Contest
Sponsored by IBM

11/11/2000

Double, Double, Toil and Trouble



The **ToxicCo** chemical company has developed a new, somewhat unstable process for electroplating metals in a hot acid bath. This process requires constant monitoring of the temperatures and voltages present in the bath. Too low a temperature or too high a voltage will cause the dissolved metals to congeal in a mass at the bottom of the vat, damaging the equipment. On the other hand, if the temperature gets too high, acid fumes may be released into the atmosphere. If the voltage goes too low, the chemical reaction is altered, and an explosion becomes possible.

A state-of-the-art, artificially-intelligent process controller is supposed not only to keep the vat temperature and voltage at safe levels, but also to continually adjust them so as to maximize the speed of the electroplating process.

Unfortunately, the software developers have reported that "there's still a few bugs in the system" and won't promise delivery of the controller for at least

another 9 months. (A few malcontents within the company are beginning to question whether the development team, headed by the nephew of the ToxicCo company president, will ever deliver a working system.)

This plant needs to start operation as soon as possible, so you have been commissioned to write a simpler control program to serve in the interim while the deluxe version is being debugged.

Write a program to read descriptions of "safe regions" within which temperature and voltage levels can range and to read current temperatures and voltage values and then to determine if the current values are safe. A safe region is described by 4 linear constraints:

- Two lower bounds of the form $aT + bV \geq c$, where T and V denote the vat temperature and voltage, respectively. a , b , and c are constants to be supplied in the program input.
- Two upper bounds of the form $aT + bV \leq c$, where the symbols are defined in the same manner.

The safe region forms a quadrilateral in the (T, V) coordinate space, and is guaranteed to be closed (i.e., the safe region does not include plus or minus infinity for either T or V).

INPUT

All input to this program is supplied from the standard input stream. The input begins with a line containing an integer value indicating the number of data sets to follow. There will be at least one data set, but no more than 500.

Each data set consists of 5 lines of text, as follows

- Line 1 will contain the a , b , and c values for the first lower bound.
- Line 2 will contain the a , b , and c values for the second lower bound.
- Line 3 will contain the a , b , and c values for the first upper bound.
- Line 4 will contain the a , b , and c values for the second upper bound.
- Line 5 will contain the current temperature and voltage.

On each line, the indicated values will be given as floating point numbers, separated from one another by one or more blanks. No line will exceed 80 characters.

OUTPUT

The output of this program will consist of a single line of text per data set, written to the standard output. Each line will contain the data set number, left-justified, a blank, and then either the word "SAFE" or the word "UNSAFE", followed by the appropriate end-of-line termination. No other output should appear.

Double, Double, Toil and Trouble

2000 Mid Atlantic Regional ACM Programming Contest

11/11/2000

Sponsored by IBM

SAMPLE INPUT

```
2
35.0 -5.0 375.0
0.0 1.0 65.0
-10.0 32.5 3000.0
55.0 -30.0 1350.0
 15.0 75.2
35.0 -5.0 375.0
0.0 1.0 65.0
-10.0 32.5 3000.0
55.0 -30.0 1350.0
 25.0 70.6
```

SAMPLE OUTPUT

```
1 UNSAFE
2 SAFE
```

Houses Divided

Houses Divided



Bill and Scott are business rivals. Each of them wishes to buy a house in Javaville, but they want to live as far away from each other as possible. Since Javaville is a relatively new town, there are no maps available yet; instead, information about homes and other buildings has been collected by word of mouth and provided to both Bill and Scott. This information consists of building addresses and distances between buildings. All of the information is consistent, although it may be incomplete or redundant.

The streets in Javaville are laid out in a rectangular grid of m east/west streets (named A, B, C, ...) and n north/south streets (numbered 0, 1, 2, ...), where m and n are each between 2 and 10. Every building (either a house or some other building, such as a post office or school) in Javaville is at the intersection of two streets, and no two buildings are located at the same intersection. Some intersections have no buildings at all. All distances are measured in terms of the smallest number of whole blocks that must be traversed north, south, east, and/or west to get from one intersection to another intersection. It is known that there are no more than 50 intersections in the entire town.

Here is some sample information that might be provided to Bill and Scott by various reliable sources:

- There are 5 east/west streets and 5 north/south streets.
- House1 is located at intersection A0
- The post office is located at intersection A4
- The school is at a distance of 4 blocks from house 1
- House2 is at a distance of 6 blocks from the post office
- The school is at a distance of 6 blocks from the post office
- House3 is at a distance of 6 blocks from the post office

From this we can see that there are two possible maps of Javaville—see Figure 1.

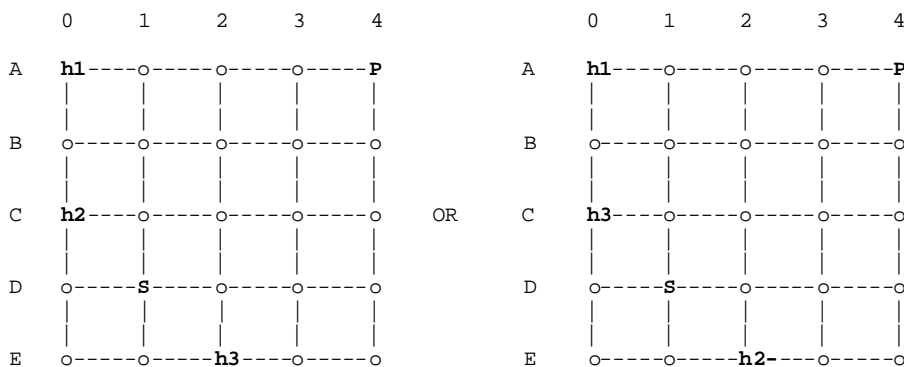


Figure 1: h1,h2,h3 = houses, P = postoffice, S = school

We see that the locations of house1, the post office, and the school are fixed, but house2 could be at either C0 or E2, and house3 could be at either C0 or E2. Clearly there is always a pair of houses separated by 6 blocks (house1 is always 6 blocks from the furthest house), but the best distance we can guarantee for any *specific pair* of houses is 4

Houses Divided

2000 Mid Atlantic Regional ACM Programming Contest

11/11/2000

Sponsored by IBM

(since house2 is always 4 blocks away from house 3). We would report this information to Bill and Scott, telling them that, even though a separation of 6 is always achievable, the safest recommendation that we are able to make for specific houses is to have one of them purchase house2 and have the other purchase house3. (We assume that Bill and Scott will consult with one another before purchasing their houses in order to guarantee that one of our recommendations is followed.)

Bill and Scott would like you to write a program that will take location and distance information about buildings and determine two quantities, D and D' . D is the minimum, over all possible valid arrangements of buildings, of the largest house separation in each arrangement. D' is the maximum value for which there exists a pair of houses i and j that are guaranteed to be separated by a distance of at least D' . In the latter case, you should list all pairs of houses that are guaranteed to be separated by at least D' blocks.

To make sure your program is working correctly, Bill and Scott would like to be able to test it on multiple data sets, so your program will be presented with several hypothetical descriptions of Javaville.

INPUT

The input will consist of one or more descriptions. Each description will begin with a line containing two positive integers, m and n , representing the number of blocks in each direction ($2 = m, n = 10$). Each description will end with a line containing the single word END in uppercase. The entire data set will end with a line containing a pair of zeros. No assignment of values for m and n will result in a town having more than 50 intersections.

The remaining lines in each data set will be in one of the following two formats:

name LOCATION r c

or

name DISTANCE d *name2*

where *name* and *name2* are strings containing only digits and lowercase alphabetic characters, each of length at most 10; r is an uppercase letter between A and J; c is a digit; and d is a positive integer. If the first five characters of a *name* are the lowercase letters "house" then it is a candidate for selection as a home for Bill or Scott; otherwise it stands for some non-residential building. In a "DISTANCE" specification, *name2* will always be the name of some building that has occurred previously in this data set as the first *name* on one of the data lines (in other words, there are no forward references to buildings in DISTANCE constraints). Each description is consistent (i.e., there is at least one way to lay out the houses and other buildings in a way consistent with the description). Each description will include information about at least two distinct houses. At most 20 distinct building names will occur in each description, and there will be at most 21 constraints (LOCATION or DISTANCE) for each description.

OUTPUT

For each description, the output will consist of the description number, followed by the maximum achievable house separation D , followed by a list of all pairs of houses with maximum guaranteed separation D' (which might be smaller than the maximum achievable separation). No pair of houses should be listed more than once. The output should be labeled exactly as shown in the sample output below, with a blank line separating the outputs for consecutive data sets. In each pair, houses should be ordered according to the first time they appear in the input; the list of pairs should be ordered in the same way, sorted by the first element of the pair, then by the second element.

Houses Divided

2000 Mid Atlantic Regional ACM Programming Contest

11/11/2000

Sponsored by IBM

SAMPLE INPUT

```
5 5
house1 LOCATION A 0
postoffice LOCATION A 4
school DISTANCE 4 house1
house2 DISTANCE 6 postoffice
school DISTANCE 6 postoffice
house3 DISTANCE 6 postoffice
END
2 3
school LOCATION A 0
house1 DISTANCE 1 school
house2 DISTANCE 1 house1
house3 DISTANCE 1 house2
house4 DISTANCE 1 house3
house5 DISTANCE 1 house4
END
0 0
```

SAMPLE OUTPUT

```
DESCRIPTION 1
Maximum guaranteed separation is 6 blocks.
Houses separated by at least 4 blocks:
house2 house3

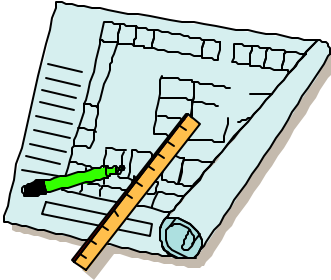
DESCRIPTION 2
Maximum guaranteed separation is 3 blocks.
Houses separated by at least 2 blocks:
house1 house3
house1 house5
house2 house4
house3 house5
```

The Computerized Contractor

2000 Mid Atlantic Regional ACM Programming Contest
Sponsored by IBM

11/11/2000

The Computerized Contractor



I.M. Crooked, general contractor, is looking for an easier way to give potential customers totally useless estimates of how much it will cost to build additions onto their houses. Although the final estimates aren't related to what the additions will cost, Mr. Crooked wants the estimates to be based on a formula so he can explain to the customers how he arrives at the estimate. You've been given the formula and must implement a program to automate his estimates.

There is a fixed base cost for adding on an addition; a cost per square foot of finished floor, ceiling, or wall; a cost per square foot of door; and a cost per square foot of window. Additionally, there are maximum widths and heights for standard doors or windows. If any window or door exceeds one of those dimensions, then the cost for that particular window or door is multiplied by a given factor.

Your program will accept a file which lists a set of prices, and then lists a set of potential additions. You must calculate and output the cost of each addition.

INPUT

line 1: <base cost>

line 2: <cost per finished square foot of floor/ceiling/wall>

line 3: <cost per square foot of door>

line 4: <max. unpenalized door width> <max. unpenalized height> <penalty multiplier>

line 5: <cost per square foot of window>

line 6: <max. unpenalized window width> <max. unpen. height> <penalty multiplier>

Each addition is a single room, and its definition will begin with a line:

room <width> <length> <height> <number of doors> <number of windows>

Immediately following that line will be one door line for each door:

door <width> <height>

Following all of the door lines will be one window line for each window:

window <width> <height>

The next addition will begin immediately following the last line belonging to the previous room.

The file will end with:

END OF INPUT

All numbers except the number of doors and the number of windows will be floating point numbers.

OUTPUT

For each room, a single line of output will show the calculated price, rounded to the nearest penny. No dollar sign will be used:

The Computerized Contractor

2000 Mid Atlantic Regional ACM Programming Contest

11/11/2000

Sponsored by IBM

SAMPLE INPUT

10000

3.45

2.10

3 7 1.5

4.5

3 5 2.1

room 25 15 10 2 1

door 3 8

door 3 7

window 4 5

END OF INPUT

SAMPLE OUTPUT

15431.95

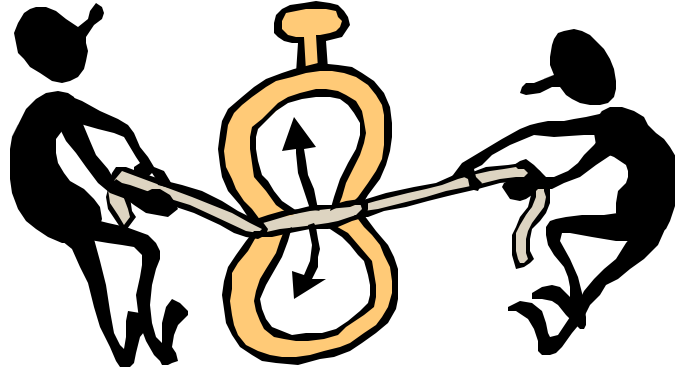
The “I’s” Have It

2000 Mid Atlantic Regional ACM Programming Contest
Sponsored by IBM

11/11/2000

The “I’s” Have It

Lon Tymago, a professor of history, is having a difficult time training his students not to use the personal pronoun “I” when they write their term papers. Most of Dr. Tymago’s students now submit their papers electronically, so the professor has hired you to write a program that will automatically scan the students’ papers, identifying occurrences of the pronoun “I”. This will eliminate one step in grading the papers.



Unfortunately, history papers often make use of the Roman numeral “I” (e.g., King George I, Pope John Paul I, World War I). Your program is not to indicate any error when it can be *clearly determined* that one of these situations has occurred.

The word “I,” used as a pronoun, frequently occurs immediately in front of a verb: “I think,” “I am,” “I did,” and so on. The same can be said of noun phrases such as “King George I” and “World War I”: “George I thinks,” “World War I is,” and so on. However, some verb forms are appropriate for only one case. It is grammatically correct to write “I think,” but it is not correct to write “George I think.” Likewise, we can say “I am,” but we cannot say “World War I am.” Your program will make use of verb forms as part of its strategy for locating the pronoun “I.” Of course, in some cases the verb form is not enough—for example, “I had” and “George I had” are both correct. Your program should identify these as indeterminate cases requiring further scrutiny by Dr. Tymago.

One other clue helps distinguish valid uses of “I” from invalid ones: the pronoun “I” rarely appears immediately before a punctuation mark. (There are exceptions, of course, as in “I, a student of history, ...,” but we will assume that these exceptions will not occur in any of the student papers.)

You will first be given a list of verb pairs. Each pair will show a form correct for the first-person singular pronoun “I” and a form correct for a third-person noun (the two forms may be the same). Following this, you will be given several essays written by Dr. Tymago’s students. Your program should print each line of text unchanged (including all white space, punctuation, etc.), preceded by a count of the number of positively identified uses of the first-person pronoun “I” and a count of the uses of “I” that are possibly incorrect. In addition, these “I”s should be marked as described below. Correct usages of “I” (as determined by the verb forms or presence of punctuation) should not be counted or marked.

INPUT

The data file will begin with a positive integer n ($= 50$) indicating the number of verb pairs, followed by n lines, each containing the first-person and third-person forms of some verb, separated by a space. Each verb form is no longer than 20 letters. Following this will be one or more student essays. Each essay will consist of at most 25 nonempty lines, each consisting of at most 60 upper and lower case letters, spaces, and punctuation marks. There are no trailing blanks on the lines. The only permitted punctuation marks are periods (“.”), commas (“,”), exclamation points (“!”), and question marks (“?”). There are no other special characters (such as parentheses, tabs, hyphens, apostrophes, etc.), and lines are broken only between words or after punctuation marks. Each student paper will be followed by a line containing only the word “END”. The entire data file will be terminated by a line containing the words “END OF DATA”. You may assume that every verb which appears following the word “I” will be one of the verb forms in the initial list, that the word “I” will never appear in the text except followed by a verb form or a punctuation mark, and that no two distinct verbs will agree on their first- and third-person forms.

The "I's" Have It

2000 Mid Atlantic Regional ACM Programming Contest

11/11/2000

Sponsored by IBM

OUTPUT

Imitate the sample output below. Each student essay should be printed out, preceded by a line containing the number of the essay in the input sequence. Each line of student text should be exactly reproduced in the output, including all leading blanks, punctuation marks, etc., but shifted 8 spaces to the right. Beneath each line of text there should be an indicator line showing a "*" below every positively identified use of the pronoun "I" and a "?" below each "I" whose meaning cannot be determined according to the rules described above. A blank indicator line should follow text lines that have no indeterminate or incorrect usages of "I". Trailing blanks are permitted in indicator lines as long as they do not cause the total line length to exceed 80 columns. In the eight spaces preceding each line of student text there should be two integers representing the number of occurrences of the pronoun "I" and the number of indeterminate uses of the word "I" (cases in which the rules are not sufficient to make a determination) for that line. A blank line should separate each student's paper from the next paper.

SAMPLE INPUT

```
10
was was
do does
have has
imitate imitates
think thinks
thought thought
am is
wonder wonders
would would
like likes
World War I was many years after the Revolutionary War. I
wonder what King George I would have thought of his grandson
losing that war. Another king, Richard I, was nicknamed
Richard the Lionhearted. I think I like him.
END
    I would like to get an A on this paper about Pope Pius I,
but I have no idea who he is.
END
END OF DATA
```

SAMPLE OUTPUT

```
STUDENT 1
  1 1 World War I was many years after the Revolutionary War. I
      ?
  0 1 wonder what King George I would have thought of his grandson
      ?
  0 0 losing that war. Another king, Richard I, was nicknamed

  2 0 Richard the Lionhearted. I think I like him.
      *      *

STUDENT 2
  0 1      I would like to get an A on this paper about Pope Pius I,
      ?
  1 0 but I have no idea who he is.
      *
```

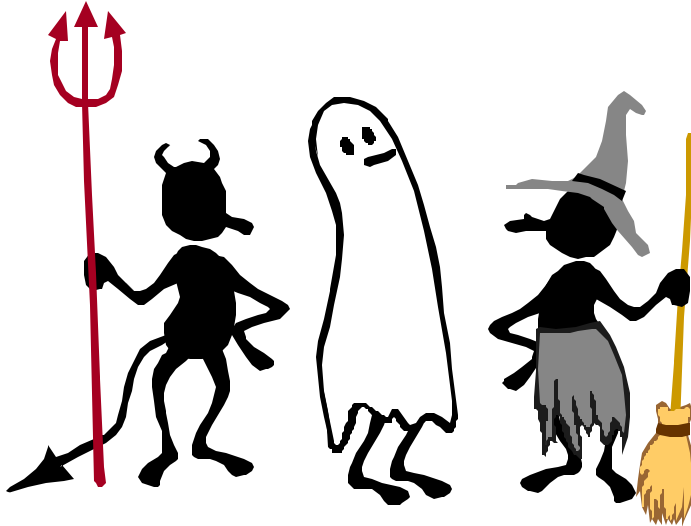
A Committee is a Beast with Many Heads and No Brain

2000 Mid Atlantic Regional ACM Programming Contest

11/11/2000

Sponsored by IBM

A Committee is a Beast with Many Heads and No Brain



In the wake of the recent controversy over the intrusive questions asked by the US Census Bureau during the 2000 census, Congress has decided that decisive action is required and so, after much debate, it has decided to ... commission a fact-finding study. The Census Bureau has been instructed to form discussion groups of people from around the country and of many different ancestries to help refine the questions for the 2010 census.

To encourage diverse responses, no two members of any discussion group can come from the same ancestry or from the same city. Furthermore, because these people will be together in small meeting

rooms for some time, smokers should not be placed in the same group with non-smokers. Finally, the caterer has suggested that serving lunch to these groups would be easier if vegetarians and non-vegetarians were not mixed together.

Write a program to read descriptions of the people participating in the fact-finding study. Your program should then assign these people to discussion groups so that no two people of the same ancestry are together in a group, no two people from the same city are together in a group, no group contains a mixture of smokers and non-smokers, and no group contains a mixture of vegetarians and omnivores (non-vegetarians). The program should also enforce limits on the maximum number of discussion groups and the maximum size of any one group. It is possible that, given such limits, assignment of all people into groups may not be possible. The program should indicate in its output whether or not a suitable assignment of people into groups is possible and, if it is possible, the membership of each group.

INPUT

All input is taken from the standard input stream. The first line of input will contain two positive integers, separated by one or more blanks. The first integer is the maximum number of discussion groups permitted. The second is the maximum number of people per group.

Each subsequent line of input describes a single person participating in the study. Each person is implicitly identified by number according to the line in which their description occurs. The first line of input, as already noted, describes the number and size of groups rather than a person. Therefore the second line of input describes person #1, the third line describes person #2, etc.

Each person's description has the form:

```
Ancestry-code City-code Smoker-code Lunch-code
```

Each of the four codes is separated from the others by one or more blanks. The Ancestry-code and City-Code are positive integers. The Smoker-code is either 'S' or 'N' (Smoker or Non-Smoker). The Lunch-code is 'V' or 'O' (Vegetarian or Omnivore). The smoker and lunch codes will always be upper-case letters. No line of input will exceed 80 characters.

The entire set of descriptions is terminated by end-of-file on the input stream. There will be at most 200 descriptions.

A Committee is a Beast with Many Heads and No Brain

2000 Mid Atlantic Regional ACM Programming Contest

11/11/2000

Sponsored by IBM

OUTPUT

All output is to the standard output stream.

If it is not possible to assign all the people to appropriate discussion groups, the program should print "NO ASSIGNMENT POSSIBLE".

If an assignment is possible, then the program should print one line of output for each non-empty discussion group. Each line of output should contain the identifying numbers of the people assigned to that group, separated from one another by a single blank character. The numbers in each line may appear in any order. At the end of this listing of group assignments, the program should print "END OF GROUPS".

The "NO ASSIGNMENT POSSIBLE" or "END OF GROUPS" messages should appear by themselves on a single line starting in the first character position of that line.

Note that, for some inputs, there may be many acceptable assignments. The program may print any acceptable assignment in such cases.

SAMPLE INPUT

```
5 3
1 1 S V
2 3 S V
3   2 N O
  1 3 S V
1 1 N O
4 4 N O
5 7 N O
```

SAMPLE OUTPUT

```
1 2
5 6 3
4
7
END OF GROUPS
```

The New FAA

2000 Mid Atlantic Regional ACM Programming Contest
Sponsored by IBM

11/11/2000

The New FAA



The FAA's most recent plan to replace every computerized component of their traffic control system has failed. They have finally decided to replace only the most important part, the collision detection system. You have been given the responsibility of writing this life-critical part of the system.

Your program receives input from an airport's radar system. The radar system is very advanced and can report the position, direction, and speed of any airplane in the vicinity. The program must determine if any plane it sees is likely to collide with any other plane in the area.

The airport's radar spins at a continuous rate, completing one revolution every 4 seconds. Data is reported about each plane as the radar faces in that direction. As each plane's data is received, the

program must determine if the new plane's course over the next 10 seconds will be too close to the course of another plane that has been seen by the radar within the past eight seconds.

The minimum requirement for proximity at this airport is that the planes always be separated by at least 2500 feet in all directions (standard three dimensional distance). If a test is made between the current plane and one seen previously, the previous plane's course is plotted by calculating where it would be at the current time if it has continued its last known direction over the time from when it was last seen to the present. For example, for a plane seen 3 seconds previously, that plane's course is plotted by calculating where it would be 3 seconds after its last observation; then the routes of the two planes over the subsequent 10 seconds is calculated. If, at any point during those 10 seconds, the planes would be within 2500 feet, the two planes are too close.

The input to the program consists of the series of radar observations. Anytime a plane's course will bring it too close to another plane's course, the program will output a warning listing the two planes. If a new plane is too close to multiple planes' flightpaths, the conflicts will be printed out from the most recently seen plane to the least recently seen.

The radar spins clockwise. Note that a plane's course may not conflict with its own course seen previously by the radar. More than one plane may be found at a particular bearing of the radar. Planes with identical bearings may be listed in any order. The input will indicate when the plane spins through 0 degrees. Planes at a bearing of 0 degrees will always be listed after this indication.

Keep only one entry for each plane. Keep no entry older than 8 seconds.

HINT: compass degrees are in a different coordinate system than standard trigonometric degrees.

Because floating point errors may be introduced by different solutions to this problem, the data in all input files will produce planes which pass either within 2490 feet or further than 2510 feet, so comparisons with 2500 feet are reliable.

The New FAA

2000 Mid Atlantic Regional ACM Programming Contest

11/11/2000

Sponsored by IBM

INPUT

Each line consists of:

<bearing> <PlaneID> <distance from radar> <elevation> <heading> <rate of ascent> <velocity>

<bearing> is [0..360)

<PlaneID> is an alphanumeric (no spaces) id for the plane, no more than 10 characters

<distance from radar> is in feet, and is in the horizontal plane only

<elevation> is in feet

<heading> is [0..360)

<rate of ascent> is in feet per second (0 is level)

<velocity> is in feet per second (in the horizontal plane)

All numbers are real-valued numbers. All numbers will be non-negative, except for rate of ascent, which may be negative for descent. However, note that the courses over the next ten seconds may produce negative coordinates, in the case of a landing plane, for instance. Your program should allow such negative coordinates for landing planes (do not force altitude to stay positive). The total length of the line will not exceed 150 characters.

A line of:

CLICK

indicates that the radar has spun through 360 degrees.

The file will terminate with

END OF INPUT

OUTPUT

Output is only produced in the case of planes' courses being too close, in which case the PlaneID of the just-spotted plane is printed followed by the ID of the previously seen plane. In the case of multiple proximity violations, print all conflicts on separate lines, with the most recently seen conflicting plane being printed first.

SAMPLE INPUT

0 planeA 1 10000 0 0 .2

75 planeB 5.79 10000 270 0 1

270 planeC 2500 10000 0 0 .2

END OF INPUT

SAMPLE OUTPUT

planeB planeA

planeC planeB

It's Time for Change

2000 Mid Atlantic Regional ACM Programming Contest
Sponsored by IBM

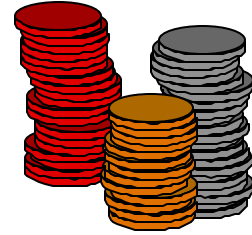
11/11/2000

It's time for change

"Louie, I bet you that there are over 200 ways to make change for a dollar."

"You're on Frankie, there are only 5 coins and its only 100 cents. Everyone knows there can only be 100 different ways."

"Oh yeah, I can prove it. Watch."



Write a program that determines the number of ways the following US coins can be combined to make amounts of money up to and including 2 dollars.

Penny - \$0.01
Nickel - \$0.05
Dime - \$0.10
Quarter - \$0.25
Half Dollar - \$0.50

INPUT

The input will consist of a sequence of lines, each line indicating the amount of change required. Input terminates with a value of 0.

OUTPUT

For each input value, a line indicating "There are X ways to make \$Y" where X is the number of ways, and Y is the input value. Each input set should have one line of output.

After all inputs sets have been run, the program will print "End of Output".

SAMPLE INPUT

```
1.00
0.51
0
```

SAMPLE OUTPUT

```
There are 292 ways to make $1.00
There are 50 ways to make $0.51
End of Output
```

Code Name: Double Omega VII

2000 Mid Atlantic Regional ACM Programming Contest
Sponsored by IBM

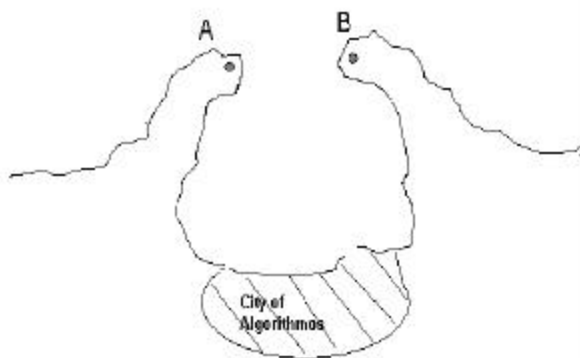
11/11/2000

Code name: Double-Omega VII

Even in ancient times when sea commerce was conducted entirely by sail or rowing, harbors were sometimes mined to discourage unwelcome visitors (smugglers, raiders, etc.). Of course, in pre-gunpowder eras, the harbor would be blocked, not with explosives. Instead, nets, chains, sunken derelict ships and other debris, lying a little below the waterline, could rip the bottom out of a ship, or at least strand a ship long enough for shore-bound archers and siege engines to come to bear.

When ships arrived on legitimate business, a local pilot would row out to meet it, board the ship, and guide it into port through the safe channels left between the underwater obstacles. The location of these safe passages was, of course, a closely-held secret that would prove invaluable to a city's enemies.

An agent for an unfriendly power has received a description of the passage through one such harbor and has been instructed to attempt the passage late at night to verify the accuracy of the information. If the information is correct, a full-scale invasion will follow.

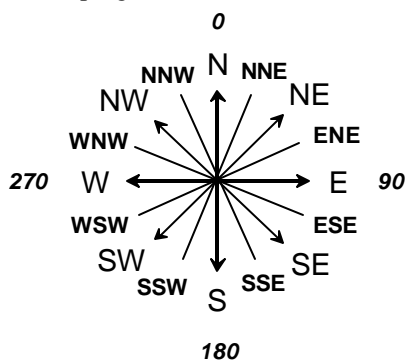


"The harbor of Algorithmos is shaped by two narrow arms of land, as shown in the accompanying map, with the open end to the north. The west and east ends of the opening are 100 stadia apart. Beacon fires, marked "A" and "B" on the map, are maintained at the ends of the two arms. At night, you will have to navigate by reference to these beacons. Starting from the midpoint of the line between those beacons, your directions are as follows..."

There follows a set of instructions of the form
"SE 10 stadia, WSW 5 stadia, S 15 stadia, ..."

The agent sets out with two sailors in a small boat with a compass and a pair of sextants (navigational instruments) that allow each sailor to measure the angle between the beacon fires and north (as indicated by the compass). They call these angles out to the agent, who must determine when they have completed a "leg" of the instructions.

Write a program that reads in the instructions for the safe path and prints, for each leg of the journey, the



angles of the two beacons at the end of that leg. The directions given in the input instructions and the measurement of angles should be conducted with respect to the conventional compass rose, as shown here. (All angles should be expressed in the range from 0 up to but not including 360, where North is 0, East is 90, NE is 45, NNE is 22.5, etc.)

You may assume that the beacons A and B lie exactly on an East-West line from one another, and are exactly 100 stadia apart. The safe path begins at the point 50 stadia from each beacon. You may assume that no safe path ever goes north of the East-West line between the beacons.

For each leg of the journey, you should print the heading (angle) of the two beacons on the compass rose as observed from the boat at end of that leg. Because of the limitations in measurement technology of the time, these outputs will be rounded to the closest degree.

Code Name: Double Omega VII

2000 Mid Atlantic Regional ACM Programming Contest

11/11/2000

Sponsored by IBM

INPUT

All input is taken from the standard input stream.

The first line of input consists of a single integer indicating the number of legs of the complete safe path. Each line after that describes a single leg of the path, giving the direction in which to travel expressed as one of the names from the compass rose ("N", "NNE", ...) followed by one or more blanks, then a distance (in stadia) given as a positive floating point number.

OUTPUT

All output goes to the standard output stream.

For each successive leg of the journey, print the heading (angle) of beacon A and beacon B on a single line, separated by a single blank. These quantities are to be rounded to the closest degree, and printed in integer format (no decimal points).

After the angles for the final leg, print the phrase "END OF PATH" on a line by itself.

SAMPLE INPUT

```
3
S 50.0
W 25.0
ESE 10.5
```

SAMPLE OUTPUT

```
315 45
333 56
327 50
END OF PATH
```