**Java program:** Prob06.java

**Input File:** Prob06.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

The goal of a word search is simple: find words in a block of letters either up, down, left, right, or diagonally. We have all played this game hundreds of times. But this isn’t your run-of-the-mill word maze. Today you will search for a word worm that bends its way in any direction through a block of letters. The worm may even overlap itself. Don’t let him get away!

Here are the properties of word worms:

* Word worms can move left, right, up, down, and diagonally.
* Word worms can overlap other word worms (even itself).

Here is an example of a word worm spelling the word LOCKHEED:

A **D E** K H E Q

B X **E H K** J R

J I **L O C** K D

R P I G N A H

T E N E F H M

J U O P L N T

As another example, the following could be used to spell the word BANANA:

B A N

**Program Input**

The first line of the file Prob06.in.txt will contain a positive integer T denoting the number of test cases that follow. Each test case will have the following input:

* The first line of each test case will contain two space separated positive integers in the form R C, where R is the number of rows in the puzzle and C is the number of columns in the puzzle.
* The next R lines will each contain C space separated capital letters – this is the puzzle board.
* The next line of each test case will contain a positive integer N denoting the number of words that follow.
* The next N lines of each test case will contain a single word in all capital letters. These are the words to search for in the puzzle.

**Example Input:**

1

6 7

A D E K H E Q

B X E H K J R

J I L O C K D

R P I G N A H

T E N E F H M

J U O P L N T

4

LOCKHEED

PLANE

JET

ENGINE

**Program Output**

Your program should print the list of words that were found in the grid, in the order they were listed in the input file. If a word could not be found, do not print it out.

**Example Output:**

LOCKHEED

JET

ENGINE

**Java program:** Prob07.java

**Input File:** Prob07.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

You are patrolling national air-space in an F-22 Raptor and your radar indicates incoming alien robot invaders. Your directive is to destroy each ship starting with the closest and ending with the furthest. Each time you destroy one ship, all remaining ships advance closer to you, but at differing rates. Class-A ships advance 10 X-units, Class-B ships advance 20 X-units, and Class-C ships advance 30 X-units.

For the purposes of this problem you are trying to protect the Y axis, so the closest ship is the one with the lowest X coordinate. In the event of a tie, you should destroy the ship with the largest Y coordinate first. Negative X coordinates are fine – it just means the aliens have invaded your airspace!

**Program Input**

The first line of the file Prob07.in.txt will contain a positive integer T denoting the number of test cases that follow. Each test case will have the following input:

* The first line of each test case will contain a positive integer N denoting the number of alien ships that follow.
* The next N lines will contain a description of an alien ship in the following format:

<ShipName>\_<Class>:<X>,<Y>

The ship name and the class of the ship will be separated by an underscore. The class and the X coordinate will be separated by a colon. The X and Y coordinates will be separated by a comma.

**Example Input:**

2

3

DOOM\_A:123,1444

TEST\_B:12,145

BOGEE\_C:52,345

13

SHIP1\_A:150,150

SHIP2\_B:200,150

SHIP3\_C:165,130

SHIP4\_A:205,135

SHIP5\_B:155,105

SHIP6\_C:195,120

SHIP7\_A:140,50

SHIP8\_B:175,70

SHIP9\_C:215,70

SHIP10\_A:145,10

SHIP11\_B:160,30

SHIP12\_C:185,35

SHIP13\_C:225,20

**Program Output**

Your program should output the data about the ships that it destroys in the order in which it destroys them. The format for each output line should be:

Destroyed Ship: <SHIPNAME>xLoc: <x>

**Example Output:**

Destroyed Ship: TEST xLoc: 12

Destroyed Ship: BOGEE xLoc: 22

Destroyed Ship: DOOM xLoc: 103

Destroyed Ship: SHIP7 xLoc: 140

Destroyed Ship: SHIP3 xLoc: 135

Destroyed Ship: SHIP5 xLoc: 115

Destroyed Ship: SHIP12 xLoc: 95

Destroyed Ship: SHIP6 xLoc: 75

Destroyed Ship: SHIP11 xLoc: 60

Destroyed Ship: SHIP9 xLoc: 35

Destroyed Ship: SHIP13 xLoc: 15

Destroyed Ship: SHIP8 xLoc: 15

Destroyed Ship: SHIP2 xLoc: 20

Destroyed Ship: SHIP10 xLoc: 45

Destroyed Ship: SHIP1 xLoc: 40

Destroyed Ship: SHIP4 xLoc: 85

**Java program:** Prob08.java

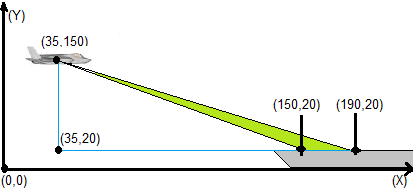
**Input File:** Prob08.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

You are in charge of directing a state-of-the-art, fifth-generation fighter jet to a safe landing aboard the USS Quest Aircraft Carrier. If the aircraft glide-slope is too steep or too shallow, you must indicate to the pilot that he or she must abort the landing and retry. Only when the glide slopes to the targeted landing zone are within tolerances are you to indicate to the pilot that he or she may commit to the landing.

You must calculate the glide slopes and determine if they are within tolerance for a safe landing. If the slope of approach is between -.8 and -1.6 (inclusive), the aircraft is clear to land – otherwise it is waved off. Because you are landing on a carrier, you must consider both the slope between the plane and the front of the landing zone as well as the slope between the plane and the end of the landing zone. Both slopes must be within tolerance for a safe landing.

****

**Program Input**

The first line of the file Prob08.in.txt will contain a positive integer T denoting the number of test cases that follow. Each test case will have the following input:

* The first line of each test case will contain a positive integer N denoting the number of aircraft sections that follow. Each aircraft section will have the following four input lines:
  + The name of the aircraft
  + The X and Y coordinates of the aircraft separated by a comma
  + The X and Y coordinates of the start of the landing zone separated by a comma
  + The X and Y coordinates of the end of the landing zone separated by a comma

**Example Input:**

2

1

ExamplePlane

35,150

150,20

190,20

2

Freebird

25,220

150,20

190,20

Lightning

75,140

110,20

150,20

**Program Output**

Your program should give instructions to the aircraft in the order they were encountered in the input file. For a given aircraft, there are two possible outputs:

<Aircraft Name>, Clear To Land!

Or

<Aircraft Name>, Abort Landing!

**Example Output:**

ExamplePlane, Clear To Land!

Freebird, Clear To Land!

Lightning, Abort Landing!