Third Annual University of Central Florida

High School Programming Tournament

Problems

Problem Name	Filename
Round and Round	SPIRAL
Say What???	SAYWHAT
Brush Up on your Latin	LATIN
Word Search	WORD
All the Marbles	MARBLES
Motocross Scoring	MOTOX
All the Right Moves	REVERSI
Room Service	SHEET
Mad Caps	MADCAPS

Call your program file <Filename>.PAS
Call your input file <Filename>.IN

For example, suppose you are solving Room Service:

Program file: SHEET.PAS
Input file: SHEET.IN

Round and Round

Filename : SPIRAL

JoAnn recently visited Walt Disney World's Magic Kingdom for the first time. The highlight of the trip was the "Mad Tea Party" ride, which was the most thrilling and dizzying experience of her 5-year-old life. In fact, she was so impressed that when she returned to kindergarten, she found that it was impossible to learn to count, because the numbers on the chalkboard appeared to be spinning and she couldn't focus on them. Fortunately, the teacher discovered that if JoAnn saw the numbers in a *spiral* format, she could see them clearly and learn them quite easily.

The Problem:

Given a natural number (an integer greater than or equal to 1), produce the sequence of natural numbers starting at 1 and ending with the given number. The sequence is to be displayed rectangularly "spiralling" outward from the first term.

The Input:

Several integers greater than or equal to 1, but less than or equal to 350, each on its own line.

The Output:

For each number, generate the appropriate rectangular "spiral," using a row-and-column format. The spiral will start in the downward direction, then proceed counterclockwise around itself (2 will be below 1, 3 to the right of 2, 4 above 3, etc.). The spiral should be left-justified, that is, there should be no spaces between the left edge of the output window and the edge of the spiral. Numbers in the same row should be separated from each other by at least one space. There should be no blank lines anywhere within the spiral, but use a single blank line to separate successive spirals.

Sample Input:

```
9
14
6
```

Sample Output:

```
7   6   5
8   1   4
9   2   3
```

```
 7   6   5
 8   1   4
 9   2   3  14
10  11  12  13
```

```
6   5
1   4
2   3
```

Say What???

Filename : SAYWHAT

Poor Durk Dummly--he's gotten himself stranded on an uncharted island, and wants to get home. The island is inhabited by many native tribes. Durk feels that if he can talk to the King, he can find a way back home. The only problem is that the King speaks a different language, and for that matter, so do each of the tribes. There are, however, tribal elders who speak more than one language. Durk needs to find a path of translators to the King.

The Problem:

Given a list of tribal inhabitants and their native tongues, find a path, if one exists, from Durk, whose native language is A, to the King, whose native language is Z, and list the path. If no such path exists, say so.

The Input:

A list of each tribal elder's name and the languages he/she speaks, each elder on his own line. Languages are denoted by a single uppercase letter. Sets of elders will be delimited by a blank line. Each set is to be processed separately.

The Output:

For each set of elders, a declaration of a valid path, if one exists, and a list of "translators" in the grapevine from Durk to the King.

Sample Input:

Salima X Z
Tunga A Y
Kroko P R M Q
Midzu B Y R
Bob B D X

Kbonga T A
Oohoora S E

Sample Output:

Path exists:
Durk speaks to Tunga
Tunga speaks to Midzu
Midzu speaks to Bob
Bob speaks to Salima
Salima speaks to King

Path does not exist.

Brush Up on Your Latin

Filename : **LATIN**

a *Latin Square* is an n xn matrix consisting of only the integers 1 through n, such that every integer appears in every row and every column exactly once. The following is a 4x4 Latin Square:

1	3	2	4
3	1	4	2
4	2	1	3
2	4	3	1

Note that there may be many others.

The Problem:

Given an integer n, produce an n xn Latin Square.

The Input:

A sequence of integers, one per line. Each integer will be between 1 and 15.

The Output:

The requested Latin Square, in a readable matrix format. A blank line should separate Latin Squares.

Sample Input:

```
3
4
```

Sample Output:

```
1 3 2
3 2 1
2 1 3

1 3 2 4
3 1 4 2
4 2 1 3
2 4 3 1
```

Word Search

Filename : WORD

Remember those word search puzzles where you have a grid of letters and you have to find a bunch of words? Wouldn't it be great if you had a program to find all those words for you? Well now's your chance to write that program and score some contest points at the same time.

The Problem

Write a program to find specified words in a given grid of letters. The grid is guaranteed to contain all of the words. *Problem constraints* :

- There will be no more than 20 words.
- The words will be less than 10 letters long.
- There will be no more than 40 columns of letters in the grid.
- There will be no more than 15 rows of letters in the grid.
- All input will be in upper case.

The Input:

The first line of input will contain two numbers R and C representing the number of rows and columns in the grid. On the following R lines will be C letters making up the grid. On the next line will be a number W representing the number of words that you must find. The next W lines will contain the words that you must search for. Note: The first row in the grid is row 1; the first column in the grid is column 1.

The Output:

For each word, print out a line in the following form:

<word> <start-row> <start-col> <end-row> <end-col>

Where:

<word> is the word that you found.
<start-row> is the row that the first letter of the word is in.
<start-col> is the column that the first letter of the word is in.
<end-row> is the row that the last letter of the word is in.
<end-col> is the column that the last letter of the word is in.

Sample Input:

```
8 9
ASWINNERN
QWTSETNOC
NERVOUSQW
FAFDFETYU
GAGSFDFGD
HGJAZZZZZ
JZCZEZZZZ
KELOPRASZ
5
WINNER
CONTEST
NERVOUS
REAGAN
NOSEFACE
```

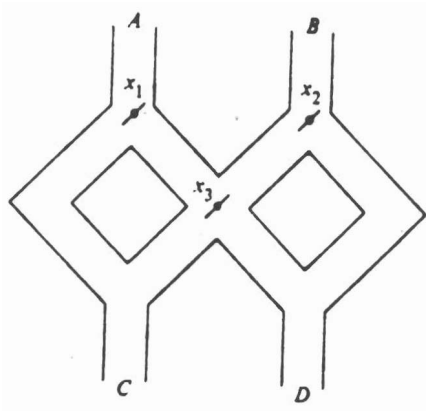
Sample Output:

```
WINNER    1  3  1  8
CONTEST   2  9  2  3
NERVOUS   3  1  3  7
REAGAN    8  6  3  1
NOSEFACE  1  9  8  2
```

All the Marbles

Filename: MARBLES

Consider the following toy:



Marbles are dropped in either slot A or slot B. If a marble encounters a lever (x_1, x_2 or x_3) it goes in the direction that the lever is slanted, and causes the lever to be slanted in the other direction. It will eventually exit the toy either through slot C or slot D. For example, in the starting configuration shown all three levers slant left. If a marble is dropped in slot B, it will first hit lever x_2. Since x_2 is slanted left, the marble will go left, and flip x_2 so that it is slanting right. The marble will then hit x_3, with the same effects. The marble will come out slot C, and the resulting state of the toy is that x_1 slants left, and both x_2 and x_3 slant right.

The Problem:

Given a string of A's and B's which represent slots that marbles are dropped into, produce a string of C's and D's corresponding to the slots the marbles would exit from. Assume at the start of each string that the levers are all three slanting left, as shown in the diagram.

The Input:

Each line will have a string consisting of nothing but capital A's and capital B's.

The Output:

for each string of A'c and B's, produce a string of nothing but capital C's and capital D's which represents the slots the marbles would exit from.

Sample Input:

```
A
B
BB
ABBA
```

Sample Output:

```
C
C
CD
CCDC
```

Motocross Scoring

Filename : MOTOX

Freddy Fripple got a call just the other day from a local businessman who is a sponsor for local motocross events: "Freddy, this is Frank Fruhart. We've decided to run things with a computer system this year, and we hear you're a pretty good programmer--with that contest at UCF and all . . . We're hoping you can write a program to keep track of scores and classifications for us." Of course, Freddy *did* need a summer job, and the program was easier to write than it seemed at first.

This could be you . . .

The Problem:

Your assignment is to keep track of rider scores and classifications for local motocross events. Each rider has a 1 or 2 digit number identifying him. Riders are categorized by experience class (C, B, A, or PRO; C is "beginner") and displacement of bike (125, 250, or 500). Each event a rider is in gives him a certain number of points. When a rider has accumulated enough points, he may move to the next experience class.

Your program's input is in two parts. The first part is a list of rider numbers, names, points already earned, and current classifications. The second part is a list of events. For each event, the results for races in different categories are given, though not all categories may be run at each event. Your program is to output a summary of each event, giving the finishers for each category in order. When all events have been processed, your program will produce a listing of all riders, how many points each now has, and what experience class each may now compete in. Points are given as follows:

Place	Points
1	20
2	10
3	5
4	2
5	1

The points needed to move to the next class also depend on the displacement of the rider's bike. The points required are as follows:

Class	125	250	500
C	25	30	40
B	60	70	90
A	95	110	150
Pro	N/A	N/A	N/A

For example, a rider in 250B who has accumulated a total of 75 points may move to 250A. Assume that a rider will not change displacements of bikes or experience classed during the execution of the program; the program will merely report what class he *could* move to at the end.

The Input:

The input will be in the following format:

n (The total number of riders)
number,name,category,points (Rider 1 starting stats)
number,name,category,points (Rider 2 starting stats)
. . .
number,name,category,points (Rider n starting stats)
eventname (Name of 1st event)
n (Number of categories run in this event)
category,1st,2nd,3rd,4th,5th (Finishers for 1st category)
category,1st,2nd,3rd,4th,5th (Finishers for 2nd category)
. . .
category,1st,2nd,3rd,4th,5th (Finishers for nth category)
eventname (Name of 2nd event)
etc.

There may be any number of events--process them until the end of the file is reached.

The Output:

Your output will consist of two parts: an event report for each event and a final report of all riders and their standings. Format your output in any readable manner.

The event report

Each event report should show the name of the event and a list of the results for each category, from 1st place down to at most 5th place. If a category was not listed in the data, print the message "DID NOT RUN" in place of the standings.

The rider report

The rider report should contain one entry per rider, listing his number, name, category, total earned points, and next possible category. Order this report by rider number.

Sample Input:

```
3
23,Joe,125C,0
54,Fred,125C,12
13,Bill,250B,50
Bithlo #1
2
125C,54,23
250B,13
Bithlo #2
1
125C,23
```

SAMPLE OUTPUT

Final standings for event Bithlo #1

Class		125	250	500
C	1	Fred(54)	DID	DID
	2	Joe(23)		
	3		NOT	NOT
	4			
	5		RUN	RUN
B	1	DID	Bill(50)	DID
	2			
	3	NOT		NOT
	4			
	5	RUN		RUN
A	1	DID	DID	DID
	2			
	3	NOT	NOT	NOT
	4			
	5	RUN	RUN	RUN
P	1	DID	DID	DID
	2			
	3	NOT	NOT	NOT
	4			
	5	RUN	RUN	RUN

Final standings for event Bithlo #2

Class		125	250	500
C	1	Joe(23)	DID	DID
	2			
	3		NOT	NOT
	4			
	5		RUN	RUN
B	1	DID	DID	DID
	2			
	3	NOT	NOT	NOT
	4			
	5	RUN	RUN	RUN
A	1	DID	DID	DID
	2			
	3	NOT	NOT	NOT
	4			
	5	RUN	RUN	RUN

P 1	DID	DID	DID
2			
3	NOT	NOT	NOT
4			
5	RUN	RUN	RUN

*** Final rider report ***

Num	Name	Current Class	Total Points	Eligible Class
---	----	-------	-----	--------
13	Bill	250B	70	250A
23	Joe	125C	30	125B
54	Fred	125C	32	125B

All the Right Moves

Filename : REVERSI

Reversi is a strategy game for two players, played on an 8 X 8 board. There are only two types of pieces: black and white. The two players, one using the black markers, the other using white, alternately place one of their markers on a square of the board until there is no legal move for the player whose turn it is. A legal move is to place a marker in any empty square such that one or more of the opposing player's markers become "surrounded." A marker or line of adjacent markers is surrounded when there is an opposing marker adjacent to it at both ends along a straight line of the board, i.e., row, column, or diagonal. The notation to represent moves uses the letters A through H to indicate columns and the numbers 1 through 8 to indicate rows.

Thus, given the following positions: (where X = Black, O = White)

```
8   .
7   .
6   .   .   O   .   .   .   .   .
5   .   .   O   X   X   X   X   .
4   .   O   X   X   .   .   .   .
3   .   .   X   .   .   .   .   .
2   .   .   .   .   .   .   .   .
1   .
    A   B   C   D   E   F   G   H
```

White has legal moves at
C2, D2, E3, E4, H5

Black has legal moves at
A4, A5, B5, B6, B7, C7

```
8   .
7   .   .   O   .   .   .   O   .
6   .   .   .   X   .   O   .   .
5   X   X   X   X   O   .   .   .
4   .   .   .   X   .   .   .   .
3   .   .   X   .   .   .   .   .
2   .   O   .   .   .   .   .   .
1   .   .   .   .   .   .   .   .
    A   B   C   D   E   F   G   H
```

White has no legal moves

Black has legal moves at
A1, F4, F5, B8, H8

The Problem:

Given a sequence of Reversi boards and a color for each board, produce a list of all legal moves for the color indicated in each position.

The Input:

Each Reversi position will consist of eight lines of eight characters each. The characters will be '.' for an empty square, 'X' for a black marker, and 'O' for a white marker. On the line immediately following the board will be either the word 'White' or the word 'Black', indicating for which color legal moves will be found in the preceding position. There will be a blank line between the color and the next board, but none following the final color.

The Output:

For each board and color pair, print one line starting with the indicated color, followed by 'has legal moves at', followed by all legal moves in Reversi notation, separated by commas. If there are no legal moves for a given position and color, print the color followed by 'has no legal moves'. There will be one line of output for each board input.

Sample Input:

```
    . . . .
  . . . . .
  OX . . .
  XO . . .


. . . . . . . .
White

. . . . O . . .
O . . . O . . .
O . . . O . . .
O . . XX . . .
O . . XO . . .
XOX . . O . .
XX . . . . O .
XOOOOOOO
Black

. . . O . . . .
. . . O . . . .
. . . O . . . .
. XXOXO . .
. . . XXX . O
. . . O . XX .
. . . . . OXO
. . . . . . . X
White
```

Sample Output:

White has legal moves at D3, C4, F5, E6
Black has legal moves at E3, B4, C4, F4, F7, A8
White has no legal moves

Room Service

Filename : SHEET

A robot valet is being programmed to turn down a sheet for its master at bedtime. The bedsheet is square, measuring 1 by 1 divads (a useless futuristic unit); it is in the middle of a square bed several divads wide. The robot sees the image of the sheet superimposed against an holographic XY-plane, with the lower left corner at (0,0) and the corner to be turned down at (1,1). The robot's programming will allow it only to place that corner within and including the confines of the space on the bed that the sheet originally occupies. Also, in case the master needs to be tucked in, the robot must know where the turned corner is. After folding the sheet down to a point specified by its master's programming, the robot is to measure the area of the figure defined by the outline of the folded square, to insure that its master will be sufficiently covered.

The Problem:

Given the position, in the robot's holographic XY-plane, of the point to which the upper right corner has been moved, determine the area of the resulting shape.

The Input:

Several sets of X and Y coordinates, representing the *new* positions of the upper right corner--guaranteed to be within and including the confines of the original square area occupied by the sheet.

The Output:

For each X and Y coordinate pair, the area of the figure formed by the placement of the upper right corner at those coordinates. Units are assumed to be square divads.

Sample Input:

```
0.5  0.5
0.8  0.654
1.0  0.75
```

Sample Output

Area = 0.875

Area = 0.954

Area = 0.875

Mad Caps

Filename : MADCAPS

It's 4:30 on a quiet Friday afternoon at Harpie and Roll Publishers. You're sitting at your desk, counting the seconds until quitting time when you leave for a weekend getaway to the mountains. Into the room bursts your boss with 500 pages of a manuscript which need to be proofread. "Correct them and get them on my desk before you leave," he says. After you recover from the deluge of expletives which fill your mind, you realize that the only thing that needs to be corrected is the capitalization of the words in the manuscript. A program on your computer could do this for you so you could leave by five...

The Problem:

Given a list of properly capitalized words followed by several lines of text, enforce proper capitalization in the text. Proper capitalization includes ensuring that each time one of the words in the list appears in the text, it is capitalized, as well as ensuring that the first word of every sentence is capitalized. *No other letters* in the sentence should be capitalized. A sentence may end with any of " . ", " ! ", or " ? " followed by any number of spaces.

The Input:

There will be one line containing the number of words in the list; this number is guaranteed to be between 1 and 20, inclusive. Following this will be the list of words--one per line--which are to correctly capitalized, followed by a blank line, followed by several lines of text. Valid punctuation marks in the text include " ! " , " . ", " ? ", " , ", and " ' ".

The Output:

Display the correctly capitalized text.

Sample Input:

```
3
shakespeare
twain
mark

great authors like William ShakeSpeare and mark twain would probably
turn over in their graves were they to know about the capitalization
of these sentences!  But why FIX them? e.e. cummings wouldn't.
```

Sample Output:

```
Great authors like william Shakespeare and Mark Twain would probably
turn over in their graves were they to know about the capitalization
of these sentences!  But why fix them? E.E. Cummings wouldn't.
```