Sixth Annual Ohio Wesleyan University Programming Contest
March 24, 2012

Rules:

1. There are six questions to be completed in four hours.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. **Do Not** include any prompts, other debugging information, or any other output except for exactly what is specified by the problem.
3. The allowed programming languages are C, C++ and Java
4. All programs will be re-compiled prior to testing with the judges' data.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed.
6. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
7. All output cases will end in a newline character. So after your program finishes output, the cursor should be on the first column of the line immediately below your last case's output.
8. All communication with the judges will be handled in the PC2 environment.
9. Judges' decisions are final. No cheating will be tolerated.

Question A: He is…the laziest man in the world

Languid von Slothful is a very lazy man. He doesn't like to walk, he doesn't even like to get up from his chair, if he can help it. The problem is that he needs to get to an important meeting across town, and needs to figure out the way to do it while expending the least amount of energy. The town has a series of bus routes, with stops on several corners. However, not all buses go where he needs to go, so Languid needs to figure out a plan. Since he's too lazy to do it himself, he's asked you to help him out.

The town is an MxN grid, which can be traversed rectilinearly (so no diagonals). Each horizontal or vertical street between locations on the grid can be traversed:

- By walking (always)
- By taking one of several bus lines (if the route of the bus includes the street)

Languid's priorities, in order, are:

1) Walk as few blocks as possible.
2) Out of all possible paths in which Languid walks the fewest blocks, choose the one that transfers between the fewest different bus lines (he'd much rather take a long indirect path than have to get up and move to a different bus).
3) If there are multiple paths that involve the fewest blocks in which Languid walks and the fewest bus transfers, any possible path is fine.

Input:

There will be several input instances. Each input will begin with 2 non-negative integers *m* and *n*, which define the number of rows and columns in the grid. (m and n are both <= 100) Values of m and n = 0 denote the end of the input. Otherwise, there will be two pairs of integers, one pair per line, *r* and *c* which denote the starting and ending row and column of the locations Languid must travel between.

Next will come a positive integer *B* (*B* <= 20), denoting the number of bus lines.
Then will be *B* bus descriptions. Each description will begin with an integer *K*, for the number of streets connected by the bus. There will follow K pairs, *r* and *c* which mark the grid locations serviced by the bus route.

Each location will be connected to at least one other location one street away, and it will be possible to get between any pair of locations serviced by the bus following the bus route (in other words, it will not be the case that one bus route has multiple disjoint components). All bus lines will be connected to each other (though possibly not directly), so there will be a path using some number of bus lines between any two locations that have a bus stop.

All row values will be between 0 and n-1, and all column values will be between 0 and m-1

Output:

For each case *i*, starting with 1, output:

Case *i*: Walking *a*, Riding *b*

Where *a* is the number of blocks needed to walk, and *b* is the number of different bus lines used.

There is a newline after each case, including the last one.

Sample Input:

```
3 6
1 0
1 5
2
3
1 1
1 2
1 3
2
1 3
1 4

3 6
1 0
1 5
3
2
1 1
1 2
2
1 3
1 4
3
1 2
1 3
2 3
0 0
```

Sample Output:

```
Case 1: Walking 2, Riding 2
Case 2: Walking 2, Riding 3
```

Question B: It Ain't Over 'Til… oh never mind, it's over.

In the coach's room during programming contests, there's often discussion about the standings. If you've been to the regionals, you know it's not uncommon for one team to finish all of the problems before the entire contest time has finished. An interesting facet of the scoring system is that it's possible for the team that finishes first in wall-clock time to not be the winner of the overall contest, due to the way penalty points are scored.

To review the scoring system: Each correct submission scores penalty points based on how many minutes have elapsed since the contest started. Each incorrect submission adds some integer *p* penalty points to the score. (In our contests, *p*=20, but it can be anything). If two teams have the same number of questions right (or both teams have all the questions right), then the team with the lower total number of penalty points wins.

Your task is to write a program that looks at the scoreboard for two teams. One team has solved the last problem, and the other team has not finished the contest yet. Given this board configuration (and the current time in the contest), does the other team still have a chance to win?

Input:

There will be multiple input instances. Each input case will start with a non-negative integer N (N <=20), for the number of questions. A value of N=0 signifies end of input.

If N > 0, there will be an integer *p*, for the penalty points. Then there will be 2 sets of N pairs of integers, one set per line:
a_1 t_1 … a_n t_n
a_i is the number of attempts the team took on question I (including the correct one). t_i is the time (in minutes) the team got the question right. All a_i and t_i values are between 0 and 1000. A value of 0 means that the team has not got the question right yet.

The first set of pairs will be for the team that got all the questions correct. The second set of pairs will be for the team still working on problems.

After the N pairs, there will be a positive integer *M*, for the current number of minutes elapsed in the contest. *M* will be at least as large as the time of the last correct submission by either team, but may be larger.

Output:

For each input case *i*, output either:
Case *i*: They still have a chance!
Or
Case *i*: It's over.

If the second team can score a tie or better by answering the rest of the questions, then they still have a chance.

There will be a newline character after each line of output, including the last.

Sample Input:

```
6
20
5 120 1 60 2 120 2 200 1 90 2 108
1 0 1 20 3 100 1 147 2 36 1 150
201

6
5
5 120 1 60 2 120 2 200 1 90 2 108
0 0 1 20 3 200 1 147 2 36 1 150
201
0
```

Sample Output:

```
Case 1: They still have a chance!
Case 2: It’s over.
```

Question C: Playing the Promotion Game

Alice stood in front of a giant chessboard. In front of her stood several pieces, staring at her menacingly. "Put on the hat", a voice whispered behind her. Turning around, she saw a bishop's miter. Shrugging, Alice put the hat on. A series of diagonal squares glowed brightly in front of her. Alice followed the path, which led to an opposing piece. The knight bowed to Alice, offering his helmet to her. "Now, you are a knight" whispered the voice. Alice put on the helmet, and the knight disappeared. A new series of squares started glowing.

Alice needs to capture all of the pieces on the chessboard, and wind up on the other side. Can you help her?

The rules:

- Every move Alice makes must capture a new piece. All pieces must be captured.
- After capturing a piece, Alice moves and captures as the new piece. She no longer moves as the old piece.
- A reminder of how chess pieces capture:
    - Pawns capture one space forward, in either diagonal
    - Bishops capture diagonally, forward or backwards, in either direction, and can move any number of spaces.
    - Rooks capture straight ahead, or side to side (not diagonally), in either direction, and can move any number of spaces.
    - Knights capture in an L-shape: Some combination of 2 squares right/left and 1 square up/down, or 2 squares up/down and 1 square right/left in either order. Knights are the only piece that can "jump over" pieces while moving without capturing them.
    - Queens moves like combinations of bishops and rooks, so have 8 possible directions of movement for any number of spaces.
    - Kings move like queens, but can only move one space
- Alice starts at the bottom edge of the board, and must finish her sequence of moves at the top edge, with all pieces captured. This means that her last move must be to capture the last piece on the top edge.
- A board configuration may not be solvable. If it is solvable, your job is to report a solution (which is a legal sequence of piece captures that reaches the goal)

Input:

The input will begin with a positive integer N, signifying the number of input instances. There will then follow N 8x8 grids of characters, separated by blank lines. Each element of the grid will be separated by a space. Each grid represents a chessboard. Each character in the grid will be either 'P' (for Pawn), 'B' (for Bishop), 'R' (for Rook), 'N' (for Knight), 'Q' (for Queen), 'K' (for King), or '-' (for an empty space). There will be exactly one non-empty space on the bottom row, and this is where Alice must start (and the piece she must start as)

Output:

For each input instance *I*, output either:
Board *i*: Alice should capture in this order: <*path*>
Where <*path*> is the sequence of hats Alice wears (see sample output for examples)
Or
Board *i*: Alice is stuck!

Sample Input:

```
3
- - - - P - - -
- - - - - - - -
- - - - - N - -
- - - - - - - -
- R - - - - - B
- - - - - - - -
- - - - - - - -
- - - - B - - -

Q - - - - - - R
- - - - - - - -
- - - - - B - -
- - - N - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
R - - - - - - -

Q - - - - - - R
- - - - - - - -
- - - - - K - -
- - - N - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
R - - - - - - -
```

Sample Output:

```
Board 1: BRBNP
Board 2: RQNBR
Board 3: Alice is stuck!
```

Question D: Squaring the Ellipse

The formula for an ellipse:

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$$

h and k are the coordinates for the center of the ellipse. a and b are the half lengths of the x and y axes, respectively, of the ellipse.

Your task is to find the vital statistics of the ellipse from an equation written in "multiplied out" form. For example:

$$x^2 + 2x + 4y^2 - 24\,y + 33 = 0$$

Is the multiplied out version of the ellipse equation:

$$\frac{(x+1)^2}{2^2} + \frac{(y-3)^2}{1^2} = 1$$

Input:

The input will begin with a positive integer N, which is the number of input instances. Each of the N instances will be on one line, and will consist of 5 non-zero integer values:

A_1 A_2 B_1 B_2 C

This represents the equation:

$$A_1x^2 + A_2\,x + B_1y^2 + B_2y + C = 0$$

Inputs will be such that the output values computed will be non-zero integers between -100 and 100

Output:

For each input instance *i*, starting with 1, output:

Ellipse i: a= <a_val> b = <b_val> h = <h_val> k = <k_val>

..where <a_val>, <b_val>, <h_val> and <k_val> are all the statistics of the ellipse given. All result values should be integers. Notice that there is a space before and after each '=' sign.

<a_val> and <b_val> will always be positive, <h_val> and <k_val> may be negative

There will be a newline after each output case, including the last one

Sample Input:

```
2
1 2 4 -24 33
9 -90 4 56 385
```

Sample Output:

```
Ellipse 1: a = 2 b = 1 h = -1 k = 3
Ellipse 2: a = 2 b = 3 h = 5 k = -7
```

Question E: Matchmaker, Matchmaker

Online matchmaker sites generally work by asking a series of questions, and then finding people with similar answers. If the answers can be expressed by integers, then one possible way to score two people's compatibility would be:

- The "difference score" of each question would be the square of the difference in the numerical values of each answer.
- The "compatibility score" is the sum of the difference scores of each question, but with the largest difference score removed. (Small scores are good, because it means the two people had similar answers to the questions)

Your task is to write a program that reads in the answers to a series of questions for two people, and computes their compatibility scores.

Input:

There will be several input instances. Each input will begin with an integer N (≤100), signifying the number of questions. A value of N = 0 signifies the end of the input

Then there will be two sets of N integers. The first set will be the answers for the first person, then the answers for the second person will be on the next line. Each value will be separated by spaces, and will be between 0 and 100

Output:

For each input value I, output:

Case i: Compatibility score of *X*.

..Where *X* is the compatibility score of the two people.

There will be a newline after each output line, including the last.

Sample Input:

```
10
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1
7
50 50 50 50 50 50 50
100 2 3 4 5 0 50
5
10 10 10 10 10
10 10 10 0 15
0
```

Sample Output:

Case 1: Compatibility score of 249.

Case 2: Compatibility score of 11154.

Case 3: Compatibility score of 25.

Question F: Scholarship Committee

You're sitting on the scholarship committee of Prestigious University (Good ol' PU). Your job is to interview applicants and decide whether to give them a scholarship or not. The students come in at arbitrary times (their visits to campus are full of tours and meetings), and all meet with you for the same amount of time. As a result, some students may be waiting in the hallway for their turn, while you meet with someone who came earlier. Each student comes in with an amount of scholarship they need to be given in order to convince them to come to PU. You have a budget you can't exceed, but would like to maximize the number of students who come. How do you allocate your scholarships?

Oh, one more thing. PU is a very selective and prestigious place, and it wouldn't do for word to get out that you give out scholarships to just anyone. To ensure the reputation of the school, you need to make sure that no student sees you give out more than 1 scholarship. In other words, from the time a student arrives and starts waiting their turn, until they're through meeting with you, you can give out at most one scholarship. (Yes, that means that if someone's waiting in the hall and sees you give out a scholarship to someone else, they won't be getting one. Try to let them down gently. Similarly, if a student arrives exactly as you're concluding a meeting with a student you're giving a scholarship to, they'll see how happy that student is, and so you can't give them a scholarship either)

Input:

There will be several input instances. The input begins with an integer N (≤ 100), denoting the number of students. A value of N=0 signals end of input.
If N is positive, there will be 2 positive integers on the next line, M and B. B is your maximum budget (in thousands of dollars), and M is the amount of time you'll be meeting with each student. B will be an integer between 0 and 1000
There will then follow N lines of 2 integers each. Each line will be information about different students. The first integer will be the time the student arrives for the meeting. The second integer will be the amount of money the student needs in scholarship (in thousands of dollars).
Times will use a 24-hour clock, and will be 4 digits long. The last 2 digits are the minutes; the first 2 digits are the hours. So 0115 is 1:15 AM, and 1459 is 2:59 PM. All input times will be between 0000 and 2359

Output:

For each input case I, starting with 1, output:
Case *i*: *k*
(where *k* is the number of students accepted)
There will be a newline after each output case, including the last one

Sample Input:

```
5
15 50
1100 10
1200 10
1205 10
1300 10
1400 15
5
30 50
1100 20
1115 10
1400 10
1335 15
1425 15
0
```

Sample Output:

```
Case 1: 4
Case 2: 3
```