Problem D

Breaking Level

Source file: n:\d\d.{c, cpp, pas}

You are playing a game with your friends up in a tall tower with *N* levels labelled 1 to *N* (1 <= *N* <= 100). You are given a object which is somewhat fragile, and you are warned that the object will break if you drop it from the "breaking level" or higher. That is, for the given object, there is a unique level *H* (1 <= *H* <= *N*) such that if the object is dropped from level *H*-1 or lower, it will not break, and if it is dropped from level *H* or higher, it will break. Your job is to determine the breaking level by dropping samples of the object from various levels of the building and observing the results.

Your friends have made the challenge a bit harder. You are given only two samples of the object. Once these have broken, there are no more objects to drop. There is one more restriction: you are given only a certain number, *D*, of opportunities to drop the object (*D* is much less than *N*). If you successfully determine the breaking level (the unique *H* defined above) before you run out of drop opportunities and using no more than the 2 samples of the object, you win the game and the unending respect and cash of your friends.

To simulate the dropping of the object from a level, your program will interact with a provided library named `level`. The library has four operations:

- `GetD` and `GetN`. Each of these should be called once at the beginning of the simulation. They return the value of *D* and *N* respectively.
- `Drop(t)`, called with an integer argument, simulates the dropping of a sample from level *t*. It returns 1 if the sample breaks; it returns 0 otherwise.
- `Decide(t)` should be called once at the end of the simulation. It indicates to the library that your program has determined that the breaking level is *t*. This routine also terminates your program.

Testing Your Program

The library reads the values of N, D, and H (in that order) from the file named level.in. You can modify the values in this file in order to test your program. After the simulation has finished, the library outputs two files. The first file, named level.out, contains the integer that your program passed to the `Decide` routine. The other file output by the library is a log file, level.log, describing the calls that your program made to the library routines.

Example

contents of level.in:

```
5 3 2
```

possible sequence of library calls:

1. `GetN` returns 5.
2. `GetD` returns 3.
3. `Drop(3)` returns 1.

4. `Drop(1)` returns 0.
5. `Drop(2)` returns 1.
6. `Decide(2)`

Instructions for Pascal Programmers

To access the library, your program must contain the statement

```
uses level;
```

For your information, here are the function and procedure declarations:

```
function GetN:integer ;
function GetD:integer ;
function Drop( level:integer ) : integer ;
procedure Decide( level:integer ) ;
```

Instructions for C/C++ Programmers

Use the instruction

```
#include "level.h"
```

in your source code. Create a project D.PRJ and add your file d.c (d.cpp) and the library level.obj to your project.
For your information, the function headers are:

```
int GetN( void ) ;
int GetD( void ) ;
int Drop( int level ) ;
void Decide( int level ) ;
```

Additional Information

Your program is not allowed to read from or write to any files.

Problem E

Fast Food

Input file: e.in
Output file: e.out
Source file: n:\e\e.{c, cpp, pas}

You are planning to open a new Veggie Vittles fast-food restaurant franchise. *N* franchises are available, and you must determine the most desirable locations.

The city is a perfect square, 10 km in linear dimension. The population density and demographics are uniform throughout the area of the city. You wish to choose your restaurant's location so that it is the Veggie Vittles that is the closest to the greatest possible fraction of the city's population.

The input may contain several test cases. The first line of each test case contains *N* (1 <= *N* <= 50); the number of Veggie Vittles to be opened. *N* lines follow, each giving the x,y coordinates of each restaurant; each coordinate value is an integer between 0 and 10. The input ends with a 0 for the value of *N*

The output from your program consists of a list for the franchise locations. Print one line for each location, in the same order as the input, giving its (x,y) coordinates followed by the percentage of the city's population for which it is the closest Veggie Vittles. Use the format given in the sample, rounding each percentage to the nearest integer. Do not worry about the details of rounding; any answer within 0.6 percentage points of the correct answer will be accepted. Output a blank line after each test case.

Sample Input

```
3
3 5
5 7
7 5
0
```

Output for Sample Input

```
Restaurant at (3,5) serves 38% of the population.
Restaurant at (5,7) serves 25% of the population.
Restaurant at (7,5) serves 38% of the population.
```

Problem F

Election Night

Input file: f.in
Output file: f.out
Source file: n:\f\f.{c, cpp, pas}

The nation of Cicoci elects its president using a winner-take-all electoral vote system. Each of the *N* states has one electoral vote, and that vote is awarded to the candidate receiving the highest popular vote in that state. The candidate with the most electoral votes is elected president. If there is a tie, no president will be elected, a constitutional crisis will ensue and the country will be plunged into civil war.

As election returns are tabulated it becomes possible to determine the outcome of the vote in a number of states. But many remain too close to call. You have been retained by Cicoci Press (CP) to compute the possible winners and losers.

The input may contain several test cases. The first line of each test case contains two positive integers: the number of states, followed by the number of candidates. Neither exceeds 100. For each state there is a line stating the number of candidates who might yet win the state, and a list of the identifiers of the candidates who might yet win the state. Candidates are identified by consecutive integers starting from 1.

For each candidate numbered *X*, in ascending order, print one of the following messages, as appropriate.

```
Candidate X will become president.
Candidate X may become president.
Candidate X will not become president.
```

Output a blank line after each test case. The input ends with a line indicating 0 states and 0 candidates.

Sample Input

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

Output for Sample Input

```
Candidate 1 will not become president.
Candidate 2 may become president.
Candidate 3 will not become president.
Candidate 4 will not become president.
Candidate 5 will not become president.
```