

NM  
i programming  
for studenter

2000



PROTECTED BY COPYRIGHT © 2000

**acm** International Collegiate  
Programming Contest

**n**extra



# Problem A

## Traffic planning

Input file: `infil.A`  
Output file: `utfil.A`

In a big town like Oslo most of the downtown streets are one-ways streets, so it can be difficult to modify the traffic patterns. Traffic manager Joan evaluates suggestions for modifying the traffic patterns, but she has problems understanding the total effect of such changes. She is afraid she will someday make a proposal that will create a situation where a street has no access.

Your task is to write a program to help Joan avoid disaster. It shall read information on the streets and their connections and then determine whether all streets are accessible from a given starting street.

### Input

First comes a description of the new traffic system. Each input line represents a street and starts with a unique number identifying it. (To make it easier for you, the streets are numbered sequentially  $1, 2, 3, \dots$ ) Then comes the name of the street (in double quotes). Following the name comes information on the streets one can reach directly from this street; they are given as a number of connections  $n$  and then the identification numbers  $r_1, r_2, \dots, r_n$  of these streets. All information concerning a street is separated by single spaces.

After the last street information line comes a line with just the number  $-1$ . Following this line comes a line with the number of the starting street.

There is no limit on the number of streets; the only limit is the size of the computer's main memory. You may assume that no street name is longer than 30 characters.

### Output

The output file shall contain the names of the streets that are *impossible* to reach from the specified starting street. These streets shall be written, one street per line, in the same sequence as they were read from the input file. If all streets are accessible, the program shall print the word "OK" instead.

#### Sample input

```
1 "Nordstrandveien" 1 5
2 "Munkerudveien" 0
3 "Oberst Rodes vei" 1 4
4 "Jordbærveien" 2 1 2
5 "Nordsetergrenda" 0
6 "Nordseter terrasse" 0
-1
4
```

#### Output for sample input

```
Oberst Rodes vei
Nordseter terrasse
```



# Problem B

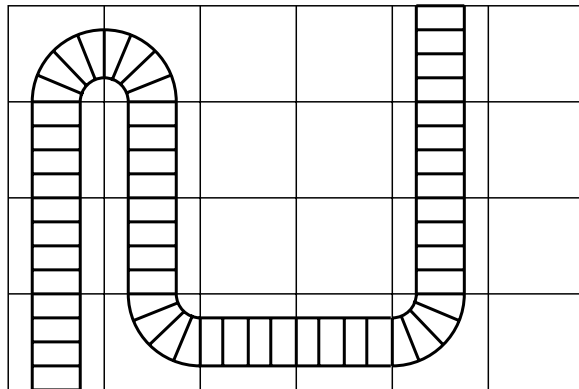
## A model railroad

Input file: `infil.B`  
Output file: `utfil.B`

John is a model railroad enthusiast. He enjoys these railroads so well that he wants to build them as long as possible. He also wants to meet other model railroad devotees and run their trains. To make it simple to transport his railroad and connect to various other railroads, each owner builds his railroad in modules. These modules are in specific sizes and designed to be easy to fit together.

John only builds rectangular modules. Every module has two connection points on the edge onto which other modules may be connected. In every module, the model railroad shall run between the two connection points, and the railroad must begin and end perpendicular to the wall at the connection points. To avoid too complex situations, John uses only straight rails and curved rails with a  $90^\circ$  bend. (A curved rail may be used to make either a left or a right turn.) A rail occupies an area of  $10 \times 10$  centimeters whether the rail is curved or straight.

John want to design his railroads so that the track is as long as possible given the number of rails he has available. To achieve this, he wants to write a computer program that calculates how many rails he may use in a module.



An solution for a  $6 \times 4$  module starting in  $(1,1)$  and ending in  $(5,4)$  when John has 15 straight and 5 curved railroad tracks.

### Input

The first input line specifies how many modules that are to be built. For every module the following information is given:

1. First comes a line stating the size of the module. This is given as two integers specifying the width and depth of the module, given as multiples of 10 centimeters. The values will always be less than or equal to 10.
2. The next line contains the connection points for the module. These connections are always along the bottom and top row, respectively. The positions are given

as the  $x$ -coordinates of the connection points and are integers in the range 1 to maximum  $x$ -coordinate.

3. The last line specifies how many straight rails and how many curved rails are available for the module.

## Output

The answer shall be how long the longest legal railroad can be. If it is impossible to build a railroad, you should state this.

### Sample input

```
2
6 4
1 5
15 5
4 4
1 4
10 1
```

### Output for sample input

```
Module 1: Longest railroad has 14 rails.
Module 2: No such railroad.
```

# Problem C

## Comment removal

Input file: `infil.C`  
Output file: `utfil.C`

Given an arbitrary, syntactically correct Pascal program, you are to compress it. (Note that you do not have to know the programming language Pascal to solve this problem; all relevant information is given below.) Here are the guidelines describing how to compress the program:

- Replace all sequences of blanks, except those within a string constant, with a single blank. Also, remove all blanks at the beginning of a line and all blanks at the end of a line.

In Pascal, strings are delimited by single quotes (“’”). Inside a string, two consecutive single quotes denote a single quote character. Examples are

```
'this is a string'  
'isn''t this fun?'
```

Strings may not cross line boundaries.

- Replace all comments with a single space. (If this space comes next to another space, the two are compressed as described above.)

In Pascal, comments are enclosed by either “(\* ... \*)” or “{ ... }”, as in

```
(* This is a comment! *)  
{ A comment with a '*' }. }  
(* A (* comment *)
```

Comments may span several lines. They may not be nested.

- Eliminate totally blank lines, i.e., lines that contain only spaces (initially, or after comment removal).
- You can assume that the source contains no TABs or other non-printing characters (except, of course, line separators).
- You can assume that no source text line is longer than 255 characters.

### Input

The input file contains a syntactically correct Pascal program, like this:

```
{_The_famous_'Hello_world'_program_}  
  
program_Hello_(output);  
_(*_no_declarations_*)  
begin  

```

(Note that space characters are shown as “\_” to make them easier to spot.)

## Output

The output should be a compressed Pascal program. For example, the program shown above should be reduced to

```
program_Hello_(output);  
begin  
WriteLn('Hello_,_,'quaint'_'_world_!');  
end.
```



Slik hjulet i figuren er satt vil teksten

TOMMY OG TIGEREN

bli kodet som

NIGGSXIAXNCA8L8H

Dessverre for medlemmene i R.Ø.L.P. har Susanne begynt å interessere seg for hva Tommy og Tigeren driver på med. Etter litt elementær etterretningsvirksomhet har hun funnet frem til det hemmelige klubbhuset og der har hun også funnet og skrevet av alle de hemmelige planene (i kodeform). Som om ikke det var ille nok er det samme firma som produserer «Super Sukrede Choko Bomber» som produserer Susannes favorittfrokostblanding «Havre og Fiber». For å gjøre produktet mer salgbart har de lagt i et tilsvarende kodehjul i frokostblandingen som det R.Ø.L.P. bruker.

Nå har Susanne alt det hun trenger for å avsløre de hemmelige planene. Hun må bare finne ut hvordan hun skal dekode hver melding. Som god hobbydetektiv har hun gjettet at bokstavkombinasjonen «RØLP» forekommer med stor sannsynlighet i hver melding. Siden du er «dataekspert» vil Susanne gjerne at du skriver et program som automatisk dekker og skriver ut en kryptert melding i klartekst. Du vet at du har dekodet en melding rett dersom du får frem bokstavkombinasjonen «RØLP». Dersom du ikke finner en dekoding som gir «RØLP» skal du gi beskjed om det. Merk at det kan finnes flere dekodinger av en melding. Da skal du gi hver mulig dekoding og så får det være opp til Susanne å avgjøre hvilken som er rett.

## Input

Den første linjen i input sier hvor mange kodemeldinger det er. For hver kodemelding er det først en linje med et tall (max 200) som sier hvor mange tegn meldingen består av. Kodemeldingen følger så på neste linje.

## Output

For hver kodemelding gi nummeret til meldingen på en egen linje. For hver mulig dekoding skal du skrive ut den dekodete teksten. Dersom det er flere måter å dekode en melding skal du skrive ut alle alternativene. Dersom det ikke er noen måte å dekode meldingen på skal du gi beskjed om det.

## Sample input

```
3
48
HGXK3JK3Z1LLKYZK3GØ3JK3QÆRK3L2X3Ø0XK3SKJ303X1RV5
21
RØLPPASSORDET ER KUEI
19
SUSANNE ER EN TAPER
```

## Output for sample input

Melding 1

Alternativ 1: BARE DE TØFFESTE AV DE KULE FÅR VÆRE MED I RØLP.

Melding 2

Alternativ 1: RØLPPASSORDET ER KUEI

Alternativ 2: Y2SwwHZZVYKLÆ4LY4RØLP

Melding 3

Ingen dekodinger



# Problem E

## A fair jury

Input file: `infil.E`

Output file: `utfil.E`

In Brutopia, a far-away country, the verdicts in court trials are determined by a jury consisting of members of the general public. Every time a trial is set to begin, a jury has to be selected, which is done as follows. First, several people are drawn randomly from the public. For each person in the pool, defense and prosecution assign a grade from 0 to 20 indicating their preference for this person. 0 means total dislike, 20 on the other hand means that this person is considered ideally suited for the jury.

Based on the grades of the two parties, the judge selects the jury. In order to ensure a fair trial, the tendencies of the jury to favour either defense or prosecution should be as balanced as possible. The jury therefore has to be chosen in a way that is satisfactory to both parties.

We will now make this more precise: given a pool of  $n$  potential jurors and two values  $d_i$  (the defense's value) and  $p_i$  (the prosecution's value) for each potential juror  $i$ , you are to select a jury of  $m$  persons. If  $\mathcal{J}$  is a subset of  $\{1, \dots, n\}$  with  $m$  elements, then  $D(\mathcal{J}) = \sum_{k \in \mathcal{J}} d_k$  and  $P(\mathcal{J}) = \sum_{k \in \mathcal{J}} p_k$  are the total values of this jury for defense and prosecution.

For an optimal jury  $\mathcal{J}$ , the value  $|D(\mathcal{J}) - P(\mathcal{J})|$  must be minimal. If there are several juries with minimal  $|D(\mathcal{J}) - P(\mathcal{J})|$ , one which maximizes  $D(\mathcal{J}) + P(\mathcal{J})$  should be selected since the jury should be as ideal as possible for both parties. If more than one jury fits these specifications, choose the one with the lowest number for the last jury member; i.e., jury "10 11 12" is preferred to "1 2 13".

You are to write a program that implements the jury selection process and chooses an optimal jury given a set of candidates.

**Note** If your solution is based on an inefficient algorithm, it may not execute in the allotted time.

### Input

The input file contains several jury selection rounds. Each round starts with a line containing the integers  $n$  and  $m$ .  $n$  is the number of candidates and  $m$  the number of jury members. These values will satisfy  $1 \leq n \leq 200, 1 \leq m \leq 20$  and, of course,  $m \leq n$ . The following  $n$  lines each contain two integers  $p_i$  and  $d_i$  for  $i = 1, \dots, n$ . A blank line separates each round from the next.

The file ends with a round that has  $n = m = 0$ .

### Output

For each round, output a line containing the number of the jury selection round ("Jury #1", "Jury #2", etc.).

On the next two lines print the values  $D(\mathcal{J})$  and  $P(\mathcal{J})$  of your jury as shown below and on another line print the numbers of the  $m$  chosen candidates in ascending order. Output a blank before each individual candidate number.

Output an empty line after each test case.

**Sample input**

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

**Output for sample input**

```
Jury #1
Best jury has value 6 for prosecution
and 4 for defence:
 2 3
```

# Problem F

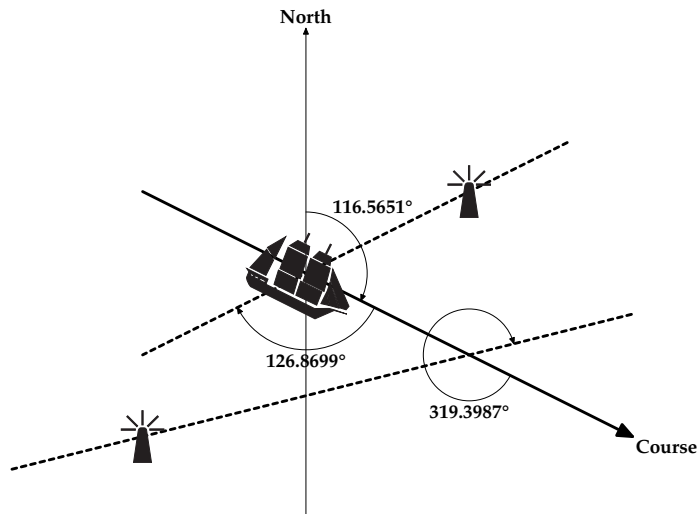
## Radio direction finder

Input file: `infil.F`

Output file: `utfil.F`

A boat with a directional antenna can determine its present position with the help of readings from local beacons (radio transmitters). Each beacon is located at a known position and emits a unique signal. When a boat detects a signal, it rotates its antenna until the signal is at maximal strength. This gives a relative bearing to the position of the beacon. Given a previous beacon reading (the time, the relative bearing, and the position of the beacon), a new beacon reading is usually sufficient to determine the boat's present position. You are to write a program to determine, when possible, boat positions from pairs of beacon readings.

For this problem, the positions of beacons and boats are relative to a rectangular coordinate system. The positive  $x$ -axis points east; the positive  $y$ -axis points north. The course is the direction of travel of the boat and is measured in degrees clockwise from north. That is, north is  $0^\circ$ , east is  $90^\circ$ , south is  $180^\circ$ , and west is  $270^\circ$ . The relative bearing of a beacon is given in degrees clockwise relative to the course of the boat. A boat's antenna cannot indicate on which side the beacon is located. A relative bearing of  $90^\circ$  means that the beacon is toward  $90^\circ$  or  $270^\circ$ .



### Input

The first line of input is an integer specifying the number of beacons (at most 30). Following that is a line for each beacon. Each of those lines begins with the beacon's name (a string of 20 or fewer alphabetical characters), the  $x$ -coordinate of its position, and the  $y$ -coordinate of its position. These fields are single-space separated.

Coming after the lines of beacon information is an integer specifying a number of boat scenarios to follow. A boat scenario consists of three lines, one for velocity and two for beacon readings.

<b>Data</b>	<b>Meaning</b>
course speed	the boat's course and the speed at which it is traveling
time <sub>1</sub> name <sub>1</sub> angle <sub>1</sub>	time of first reading, name of first beacon, relative bearing of first beacon
time <sub>2</sub> name <sub>2</sub> angle <sub>2</sub>	time of second reading, name of second beacon, relative bearing of second beacon

All times are given in minutes since midnight measured over a single 24-hour clock. The speed is the distance (in units matching those of the rectangular coordinate system) per minute. The second line of the scenario gives the first beacon reading as the time of the reading (an integer), the name of the beacon, and the angle of the reading as measured from the boat's course. These 3 fields have single space separators. The third line gives the second beacon reading. The time for that reading will always be at least as large as the time for the first reading.

### Output

For each scenario, your program should print the scenario number (Scenario 1, Scenario 2, etc.) and a message indicating the position (rounded to 2 decimal places) of the boat as of the time of the *second* reading. If it is impossible to determine the position of the boat, the message should say

Position cannot be determined

### Sample input

```
4
First 2.0 4.0
Second 6.0 2.0
Third 6.0 7.0
Fourth 10.0 5.0
2
0.0 1.0
1 First 270.0
2 Fourth 90.0
116.5651 2.2361
4 Third 126.8699
5 First 319.3987
```

### Output for sample input

```
Scenario 1: Position cannot be determined
Scenario 2: Position is (6.00,5.00)
```

# Problem G

## Returning books

Input file: `infil.G`  
Output file: `utfil.G`

I mean your *borrowers of books*—those mutilators of collections, spoilers of the symmetry of shelves, and creators of odd volumes.

— Charles Lamb  
*Essays of Elia* (1823)  
“The two races of men”

Like Mr. Lamb, librarians have their problems with borrowers too. People don't put books back where they should. Instead, returned books are kept at the main desk until a librarian is free to replace them in the right places on the shelves. Even for librarians, putting the right book in the right place can be very time-consuming. But since many libraries are now computerized, you can write a program to help.

When a borrower takes out or returns a book, the computer keeps a record of the title. Periodically, the librarian will ask your program for a list of books that have been returned so the books can be placed in their correct place on the shelves. Before they are put on the shelves, the returned books are sorted by author first and then title using the ASCII character set. Your program should output the list of returned books in the same order as they should appear on the shelves. For each book, your program should tell the librarian which book (including those previously shelved) is already on the shelf immediately before which the returned book should go.

### Input

First, the stock of the library will be listed, one book per line, in no particular order. Initially, they are all on the shelves. No two books have the same title. The format of each line will be:

`"title" by "author"`

The end of the stock listing will be marked by a line containing only one word:

`END`

Following the stock list will be a series of records of books borrowed and returned, and requests from the librarians for assistance in restocking the shelves. Each record will appear on a single line, in one of the following formats:

`BORROW "title"`  
`RETURN "title"`  
`SHELVE`

The list will be terminated by a line containing only the word

`END`

## Output

Each time the SHELVE command appears, your program should output a series of instructions for the librarian, one per line, in the format:

```
Put "title1" after "title2"
```

or, for the special case of the book being the first in the collection:

```
Put "title" first
```

After each set of instructions for each SHELVE, output a line containing only one word:

```
END
```

## Assumptions

1. A title is at most 80 characters long.
2. An author name is at most 80 characters long.
3. A title will not contain the double quote characters ("").

## Sample input

```
"The Canterbury tales" by "Chaucer, G."  
"Algorithms" by "Sedgwick, R."  
"The C programming language" by "Kernighan, R"  
END  
BORROW "The Canterbury tales"  
BORROW "Algorithms"  
BORROW "The C programming language"  
RETURN "Algorithms"  
RETURN "The C programming language"  
SHELVE  
RETURN "The Canterbury tales"  
SHELVE  
END
```

## Output for sample input

```
Put "The C programming language" first  
Put "Algorithms" after "The C programming language"  
END  
Put "The Canterbury tales" first  
END
```