Task Overview Sheet

TASK		MAINTAIN	CODE	REVERSE
Time limit per test case		1 second CPU	2 seconds CPU	N/A
Memory limit		64 MB	64 MB	N/A
Compiler options	**C**	`-pipe -O2 -lm`		N/A
	C++	`-pipe -include /usr/include/stdlib.h -O2 -lm`		N/A
	Pascal	`-So -O1 -XS`		N/A
Number of tests		20	20	16
Maximum points per test		5	5	7
Maximum total points		100	100	100

Program header comment when using C		`/*`
		`TASK: maintain`
		`LANG: C`
		`*/`
			`/*`
			`TASK: code`
			`LANG: C`
			`*/`
				N/A

Program header comment when using C++		`/*`
		`TASK: maintain`
		`LANG: C++`
		`*/`
			`/*`
			`TASK: code`
			`LANG: C++`
			`*/`
				N/A

Program header comment when using Pascal		`{`
		`TASK: maintain`
		`LANG: PASCAL`
		`}`
			`{`
			`TASK: code`
			`LANG: PASCAL`
			`}`
				N/A

Solution acceptance rule		Example input is solved correctly.	Example input is solved correctly.	The file is in the format specified.

Task Overview Sheet

TASK		MEDIAN	HOME	BALANCE
Time limit per test		1 sec	1 sec	N/A
Memory limit		64 MB	64 MB	N/A
Compiler options	**C**	`-pipe –O2 –lm`		N/A
	C++	`-pipe –include /usr/include/stdlib.h –O2 -lm`		N/A
	Pascal	`-So –O1 –XS`		N/A
Number of tests		20	20	10
Maximum points per test		5	5	10
Maximum total points		100	100	100

Program header comment when using C		`/*`
		`TASK: median`
		`LANG: C`
		`*/`
			`/*`
			`TASK: home`
			`LANG: C`
			`*/`
				N/A

Program header comment when using C++		`/*`
		`TASK: median`
		`LANG: C++`
		`*/`
			`/*`
			`TASK: home`
			`LANG: C++`
			`*/`
				N/A

Program header comment when using Pascal		`{`
		`TASK: median`
		`LANG: PASCAL`
		`}`
			`{`
			`TASK: home`
			`LANG: PASCAL`
			`}`
				N/A

Solution acceptance rule		Example input is solved correctly.	Example input is solved correctly.	The file is in format specified.

Task Overview Sheet

TASK		GUESS	ROBOTS	BOUNDARY
Time limit per test case		1 second CPU	2 seconds CPU	1 second CPU
Memory limit		64 MB	64 MB	64 MB
Compiler options	**C**	`-pipe -O2 -lm`		
	C++	`-pipe -include /usr/include/stdlib.h -O2 -lm`		
	Pascal	`-So -O1 -XS`		
Number of tests		20	20	25
Maximum points per test		5	5	4
Maximum total points		100	100	100

Program header comment when using C		`/*`
		`TASK: guess`
		`LANG: C`
		`*/`
			`/*`
			`TASK: robots`
			`LANG: C`
			`*/`
				`/*`
				`TASK: boundary`
				`LANG: C`
				`*/`

Program header comment when using C++		`/*`
		`TASK: guess`
		`LANG: C++`
		`*/`
			`/*`
			`TASK: robots`
			`LANG: C++`
			`*/`
				`/*`
				`TASK: boundary`
				`LANG: C++`
				`*/`

Program header comment when using Pascal		`{`
		`TASK: guess`
		`LANG: PASCAL`
		`}`
			`{`
			`TASK: robots`
			`LANG: PASCAL`
			`}`
				`{`
				`TASK: boundary`
				`LANG: PASCAL`
				`}`

Solution acceptance rule		Example input is solved correctly.	Example input is solved correctly.	Example input is solved correctly.

Trail Maintenance (interactive task)

TASK

Farmer John's cows wish to travel freely among the N ($1 \le N \le 200$) fields (numbered 1…N) on the farm, even though the fields are separated by forest. The cows wish to maintain trails between pairs of fields so that they can travel from any field to any other field using the maintained trails. Cows may travel along a maintained trail in either direction.

The cows do not build trails. Instead, they maintain wild animal trails that they have discovered. On any week, they can choose to maintain any or all of the wild animal trails they know about.

Always curious, the cows discover one new wild animal trail at the beginning of each week. They must then decide the set of trails to maintain for that week so that they can travel from any field to any other field. Cows can only use trails which they are currently maintaining.

The cows always want to minimize the total length of trail they must maintain. The cows can choose to maintain any subset of the wild animal trails they know about, regardless of which trails were maintained the previous week.

Wild animal trails (even when maintained) are never straight. Two trails that connect the same two fields might have different lengths. While two trails might cross, cows are so focused, they refuse to switch trails except when they are in a field.

At the beginning of each week, the cows will describe the wild animal trail they discovered. Your program must then output the minimum total length of trail the cows must maintain that week so that they can travel from any field to any other field, if there exists such a set of trails.

Input: *standard input*

- The first line of input contains two space-separated integers, N and W. W is the number of weeks the program will cover ($1 \le W \le 6000$).
- For each week, read a single line containing the wild animal trail that was discovered. This line contains three space-separated integers: the endpoints (field numbers) and the integer length of that trail (1…10000). No wild animal trail has the same field as both of its endpoints.

Output: *standard output*

Immediately after your program learns about the newly discovered wild animal trail, it should output a single line with the minimum total length of trail the cows must maintain so that they can travel from any field to any other field. If no set of trails allows the cows to travel from any field to any other field, output "`-1`".

Your program must exit after outputting the answer for the last week.

Example exchange:

Input	*Output*	*Explanation*
`4 6`		
`1 2 10`		
	`-1`	No trail connects 4 to the rest of the fields.
`1 3 8`		
	`-1`	No trail connects 4 to the rest of the fields.
`3 2 3`		
	`-1`	No trail connects 4 to the rest of the fields.
`1 4 3`		
	`14`	Maintain 1 4 3, 1 3 8, and 3 2 3.
`1 3 6`		
	`12`	Maintain 1 4 3, 1 3 6, and 3 2 3.
`2 1 2`		
	`8`	Maintain 1 4 3, 2 1 2, and 3 2 3.
	program exit	

CONSTRAINTS

Running time	1 second of CPU
Memory	64 MB

SCORING

You will receive full points on each test case for which your program produces the correct output. No partial credit will be given on any test case.

Test Data for Trail Maintenance

Algorithms:

Algorithm 1: After each year, compute the minimum spanning tree (MST). When computing the next year's MST, only consider the trails within the previous year's MST and the trail added.

This algorithm takes O(N Y log N). It is expected to receive full points.

Algorithm 2: Use a true incremental minimum spanning tree. Each year, determine the path between the endpoints of the new trail and find the maximum length trail in that path. If the length of this maximum trail is greater than the new trail, delete that trail and add the new one. Otherwise, ignore the new trail.

This algorithm takes O(N Y). It is expected to receive full points.

Algorithm 3: Recompute the MST after each year, considering all trails ever seen. Only consider the best trail between any pair of nodes. Use Prim's or Kruskal's algorithm to compute the MST.

This algorithm takes $O(Y^2 \log N)$ time. It is expected to receive around 60% of the points.

Algorithm 4: Recompute the MST after each year, considering all trails ever seen. Use Prim's or Kruskal's algorithm to compute the MST.

This algorithm takes $O(Y^2 \log Y)$ time. It is expected to receive around 50% of the points.

Test Data

Test #	Points	N	W
1	5	10	30
2	5	20	40
3	5	50	70
4	5	80	100
5	5	120	200
6	5	200	300
7	5	200	500
8	5	200	1000
9	5	200	1500
10	5	200	2000
11	5	40	3000
12	5	50	4000
13	5	200	4000
14	5	20	6000
15	5	40	6000
16	5	60	6000
17	5	80	6000
18	5	100	6000
19	5	200	6000
20	5	200	6000

Comparing Code

TASK

Racine Business Networks (RBN) has taken the Heuristic Algorithm Languages (HAL) company to court, claiming that HAL has taken source code from RBN UNIX™ and contributed it to the open-source operating system HALnix.

RBN and HAL both use a programming language with one statement per line, each of the form: `STOREA = STOREB + STOREC` where `STOREA`, `STOREB`, and `STOREC` are variable names. In particular, the first variable name starts in the first column and is followed by a space, an equals sign, a space, a second variable name, a space, the addition symbol, a space, and a third variable name. The same variable name might appear more than one time on a single line. Variable names consist of 1…8 uppercase ASCII letters ('A'…'Z').

RBN claims that HAL copied a consecutive sequence of lines directly from RBN's source code, making only minor modifications:

- RBN claims that HAL changed some of the variable names in order to disguise their crime. That is, HAL took a series of lines from RBN's program and, for each variable in it, changed all occurrences of that variable to a new variable name, although the new variable name might be the same as the original. Of course, no two variables were changed to the same new variable name.
- RBN also claims HAL might have changed the order of the right-hand side of some lines: `STOREA = STOREB + STOREC` might have been changed to `STOREA = STOREC + STOREB`.
- RBN claims that HAL did not change the order of the lines of RBN's source code.

Given source code for programs from RBN and HAL, find the longest consecutive sequence of lines from HAL's program that could have come from a consecutive sequence of lines from RBN's program using the modifications above. Note that the sequences of lines from the two programs do not have to start at the same line number in both files.

Input: `code.in`

- The first line of input contains two space-separated integers, R and H ($1 \le R \le 1000$; $1 \le H \le 1000$). R is the number of lines of source code in RBN's program; H is the number of lines of source code in HAL's program.
- The next R lines contain RBN's program.
- The next H lines contain HAL's program.

Example input:

```
4 3
RA = RB + RC
RC = D + RE
RF = RF + RJ
RE = RF + RF
HD = HE + HF
HM = HN + D
HN = HA + HB
```

Output: `code.out`

The output file should contain a single line with a single integer that is the length of the longest consecutive sequence of lines that HAL might have copied from RBN and transformed.

Example output:

```
2
```

Lines 1-2 of RBN's program are the same as lines 2-3 of HAL's program, if the following variable name substitutions are performed on RBN's program: RA → HM, RB → D, RC → HN, D → HA, RE → HB. There is no matching with three or more lines.

CONSTRAINTS

Running time	2 seconds of CPU
Memory	64 MB

SCORING

You will receive full points on each test case for which your program produces a correct output file. No partial credit will be given on any test case.

Test Data for Comparing Code

Algorithms:

This problem can be broken into two pieces: determining if there exists a mapping that takes a particular set of lines from the first program to a particular set of lines from the second (call this the inner algorithm), and determining the best such range (call this the outer algorithm). The efficiency of the outer algorithm depends on the functions provided by the inner one.

Given two sets of lines, you can compute if there is a variable mapping taking the entire first set to the other fairly efficiently. Each line is an equation. The corresponding equation of an equation is the equation on the same line in the other program. For each variable mentioned in each program, look at all the lines in which it appears in that program. If it ever appears on the left-hand side, then the variable must be mapped to the value on the left-hand side in the corresponding equation in the other program. If it appears on the right-hand side twice in the some equation, it must match to the value on the right-hand side in the corresponding equation in the other program (and the corresponding equation must use the same variable). If a variable appears only on the right-hand sides of equations in one program, then the corresponding right-hand sides of all occurrences of that variable must share a common variable. Clearly, if a variable must map to two different variables, then no mapping exists. If the variable mapping of some variable X is known to be Y, then if any occurrence of X does not have Y in the corresponding equation, then no mapping exists. Note that variables from both programs must be tested.

All of these conditions are trivially necessary, but it may be less clear that they are sufficient. If a variable X must map to another variable Y by the conditions above, then it is checked by the conditions above to ensure that every occurrence of variable X can be mapped to the variable Y. Note that variable X is known to map to variable Y if and only if variable Y is known to map to variable X. If variable X can match to either variable Y or to variable Z ($Y \neq Z$), then there are several situations: the mapping for neither variable Y nor variable Z is known, the mapping for either variable Y or variable Z is known, or the mappings for both variable Y and variable Z are known.

Neither known: If neither mapping is known, then every time variables Y or Z appear, they must appear on the right-hand side of an equation. Furthermore, the right-hand side of the corresponding equation must be X and some other variable W. Thus, the variables can map either way.

One known: If variable X can map to either Y or Z, then there is some line that causes variable X and some variable W to possibly map to either Y or Z. Without loss of generality, presume the mapping of Y is known. Since Y cannot map to X, it must map to W, or the conditions would have rejected the program set. Since the mapping of

variable Z is not known, then it must not appear in any other places or always appear paired with X and W. Thus, X can be mapped to Z, and vice-versa.

Both known: If variable X can map to either Y or Z, then there is some line that causes variable X and some variable W to possibly map to either Y or Z. If the mapping of Y and Z are known, then they must map to that variable W, which is both not legal and will be caught by the conditions above.

For the purposes of analysis, assume, without loss of generality, $R \leq H$.

Algorithm 1: For each line offset from –R to R, start at the first pair of lines that differ by the desired offset. Attempt to add pairs of lines to the end of each subprogram. If a pair of lines cannot be added (as determined an incremental algorithm that tests the conditions above), delete pairs of lines from the start of the subprograms until the pair of lines can be added or until the subprograms become empty.

This algorithm is O(R H), presuming the conditions are incrementally checked in O(1) time. It requires a method to incrementally update the conditions both by adding lines to the end of the subprograms of consideration and delete lines from the beginning of the subprograms.

This algorithm is expected to receive full points.

Algorithm 2: Start at a pair of lines from each program. Attempt to add pairs of lines to the end of each subprogram until the addition causes a conflict.

This algorithm is $O(R^2 H)$, presuming the conditions are incrementally checked in O(1) time. It requires a method to incrementally update the conditions both by adding lines to the end of the subprograms of consideration.

This algorithm is expected to receive about 65% of the points.

Algorithm 3: For each pair of starting locations, consider each possible program pairing with that pair of starting location, doing binary search to find the optimal length.

This algorithm is $O(R^2 H \log R)$ time, presuming the time to check the existence of a mapping can be done in time linear to the number of lines. It does not require support for any incremental operations.

This algorithm is expected to receive about 55% of the points.

Algorithm 4: Rather than do the O(1) mapping checker, use maximum-weighted bipartite matching. Add an edge between every pair of variables names with weight equal to the number of pairings satisfied by their pairing. If the maximum weight bipartite matching

is 3·k, where k is the number of lines in the subprograms of consideration, then there exists a mapping.

The efficiency of this algorithm varies based on the outer algorithm used. This algorithm is expected to receive about 45% of the points, although this will vary depending on algorithm and implementation.

Algorithm 5: Attempt to construct a variable mapping by constrained search of all possible mappings.

The efficiency of this algorithm varies dramatically based on the exact algorithm used. In general, this class of algorithms is expected to receive about 30% of the points.

Test Data

Test #	Points	R	H	Answer
1	5	4	4	4
2	5	5	4	4
3	5	9	6	0
4	5	5	4	4
5	5	12	3	3
6	5	10	1	1
7	5	200	10	2
8	5	90	70	43
9	5	125	110	40
10	5	180	170	60
11	5	230	210	132
12	5	354	318	178
13	5	403	364	198
14	5	465	438	203
15	5	523	497	211
16	5	678	659	212
17	5	804	787	256
18	5	904	909	274
19	5	1000	1000	306
20	5	1000	1000	98

Reverse (output-only task)

TASK

Consider a Two-Operation Machine (TOM for short) with nine registers, numbered 1…9. Each register stores a non-negative integer in the range 0…1000. The machine has two operations:

`S i j`	Store one plus the value of register i into register j. Note that i may equal j.
`P i`	Print the value of register i.

A TOM program includes a set of initial values for the registers and a sequence of operations. Given an integer N ($0 \le N \le 255$), generate a TOM program that prints the decreasing sequence of integers N, N-1, N-2, …, 0. The maximum number of consecutive S-operations should be as small as possible.

Example of a TOM program and its execution for N=2:

Operation	New Register Values									Printed Value
	1	2	3	4	5	6	7	8	9	
Initial values	0	2	0	0	0	0	0	0	0	
`P 2`	0	2	0	0	0	0	0	0	0	2
`S 1 3`	0	2	1	0	0	0	0	0	0	
`P 3`	0	2	1	0	0	0	0	0	0	1
`P 1`	0	2	1	0	0	0	0	0	0	0

Input cases are numbered 1 through 16 and are available via the contest server.

Input:

- The first line of the input file contains K, an integer specifying the input case number.
- The second line of input contains N.

Example input:

```
1
2
```

Output:

The first line of output should be the string "`FILE reverse K`", where K is the case number.

The second line of output should contain nine space-separated values representing the desired initial values of the registers, in order (register 1, then register 2, etc.).

The rest of the output file should contain the ordered list of operations to be performed, one per line. Thus, the third line contains the first operation to perform, and so on. The last line of the file should be the one that prints 0. Each line should be a valid operation. The instructions should be formatted as in the example output.

Example output #1 (partial points):

```
FILE reverse 1
0 2 0 0 0 0 0 0 0
P 2
S 1 3
P 3
P 1
```

Example output #2 (full points):

```
FILE reverse 1
0 2 1 0 0 0 0 0 0
P 2
P 3
P 1
```

SCORING

Scoring of each test case will be based on correctness and optimality of the TOM program given.

Correctness: 20%

A TOM program is correct if it performs no more than 131 consecutive S-operations and the sequence of values printed by it is correct (contains exactly N+1 integers, starting at N and ending at 0). If any S-operation causes a register to overflow, the TOM program is considered incorrect.

Optimality: 80%

Optimality of a correct TOM program is measured by the maximum number of consecutive S-operations in the program, which should be as small as possible. Scoring will be based on the difference between your TOM program and the best known TOM program.

Test Data for Reverse

Algorithms:

Algorithm 1: Assign each of the first eight register one of the eight bits of 256. Set register 9 to 0. The goal of each register is to store the next number to be outputted whose last bit that is non-zero is the bit assigned to that register. For example, the starting values are: 128, 192, 224, 240, 248, 252, 254, and 255 (and 0, of course). As soon as the value 248 is printed out, it's goal becomes 232. Select a limit on the number of consecutive S-operations. After each P-operation, perform this number of S-operations, getting the register closer to their goal values. For each S-operation, of the registers which are not yet equal to their goal, select the goal which is largest and use the S-operation to get that register closer. If, at any time during the entire program, the goal is not ready by the time it is needed, increase the limit of S-operations and start over.

This algorithm scores 85% of the points.

Algorithm 2: Consider the case of trying to solve each input with only one S-operation. Clearly, register 1 might as well as be initialized to N. The register 2 can be N-2. After printing out N, one S-operation turns register 1 to N-1. Register 3 can be N-5. After printing N-2, S 3 1 makes register 1 N-4. After printing out N-2, S 1 2 turns register 2 into N-3, the next value to output. Continuing this through all the registers, 44 is the largest value of N which can be solved in only one S-operation.

This algorithm scores 90% of the points, if extended to dealing with multiple operations.

Algorithm 3: Perform algorithm 1, but have a fixed limit of 5 operations, instead of determining of the limit for each test case.

This algorithm scores 64% of the points.

Algorithm 4: Initialize the first register to zero, the next seven registers to $\lceil k\ N / 8 \rceil$. Reserve the last for work, but set its initial value to N. At each step, pick the register that is closest to the next value to be printed (the goal), without being over. If it is not equal to the goal, do an S-operation to store its value plus one into the work register. Keep doing S-operations, adding one to the work register, until it reaches the goal value.

This algorithm scores about 50% of the points.

The efficiency of this algorithm varies dramatically based on the exact algorithm used. In general, this class of algorithms is expected to receive about 30% of the points.

Test Data

Test #	Points	N	S limit
1	6	2	0
2	6	8	0
3	6	12	1
4	6	14	1
5	6	26	1
6	6	33	1
7	6	44	1
8	6	50	2
9	6	75	2
10	6	97	~~3~~2
11	6	112	3
12	6	140	4
13	7	173	4
14	7	200	4
15	7	240	4
16	7	255	4

Note that for test case 10, one student gave a better answer than our previously best-known answer.

Median Difference

TASK

Write a program that reads a set of N (1 <= N <= 20,000) distinct integers (range: 1…250,000) and selects a set of three different values from that set such that the absolute value of the difference between the median and the mean is maximized.

Recall that the median of three numbers is the middle number when the numbers are sorted. The mean of three numbers is the average: the sum divided by 3. For the set {1, 4, 10}, the median is 4 and the mean is 5.0 ((1+4+10)/3).

Input: `median.in`

The first line of the input file contains a single integer, N.

The next N lines of the input file each contain a single integer in the input set.

Example input:

```
5
100
234
430
120
489
```

Output: `median.out`

The output file should contain the three unique integers that maximize the absolute value of the difference between their median and their mean. The integers should be listed in any order on three lines, one per line.

If there are multiple sets of three integers that have the maximum absolute difference between their median and their mean, list any one of those sets.

Example output:

```
489
100
120
```

CONSTRAINTS

Running time	1 second of CPU
Memory	64 MB

SCORING

You will receive full points on each test case for which your program produces a correct output file. No partial credit will be given on any test case.

Driving Home

TASK

You have just purchased a shiny new Segway human transporter to drive around town. Unfortunately, you are still learning how to drive it. You can steer it very well, but your control over speed is unpredictable.

When you have control of your Segway's control, your Segway takes exactly one minute to travel one block. When you lose control of your Segway's speed, however, your Segway takes you two blocks in a single minute.

You will always know at the beginning of a minute whether or not you will lose control of your Segway's speed. Also, at the start of any minute you can steer the Segway in any direction you choose. You cannot change direction except at the start of a minute. You are able to change direction at the start of each minute.

Write a program to help you get home. Your city is laid out as a perfect grid, with roads running north-south and east-west. At the start of each minute you find out whether or not you will have speed control for that minute. After learning the status of your speed control, determine which direction to steer your Segway. Minimize the number of minutes it takes you to get home.

Input: *standard input*

The first line of input contains two space-separated integers X and Y
(-100 <= X <= 100; -100 <= Y <= 100). X is the number of streets you are east of home (a negative X means you are west of home); Y is the number streets you are north of home (a negative Y means you are south of home).

For each minute, you will get a single line containing a single integer: 1 or 2. 1 means you have control of your Segway's speed and will go one block in a direction you choose. 2 means you do not have control of your Segway's speed and will go two blocks, also in any direction you choose.

Output: *standard output*

After learning the status of your speed, you must output a single character on a single line specifying the direction you want to direct your Segway. The character must be 'E' for east, 'W' for west, 'N' for north, or 'S' for south.

You must get home in 300 minutes or less to get any points. This will be possible for all input cases.

Your program must exit the first time you reach home.

Example exchange:

Input	*Output*	*Location*
3 -2		3 -2
1		
	W	2 –2
2		
	N	2 0
1		
	W	1 0
2		
	W	-1 0
1		
	E	0 0
	program exit	

CONSTRAINTS

Running time	1 second of CPU
Memory	64 MB

SCORING

Correctness: 20%

If you get home by the end of the 300th minute and properly exit, you will get at least 20% of the points for that test case. You must not output any invalid directions.

Optimality: 80%

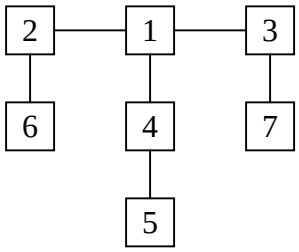
Optimality of the program is measured by the number of minutes you take to get home on your Segway. Scoring will be based on the difference between your program's output and the optimal output.

Balancing Act

TASK

Consider a tree T with N ($1 \leq N \leq 20{,}000$) nodes numbered 1…N. Deleting any node from the tree yields a **forest**: a collection of one or more trees. Define the **balance** of a node to be the size of the largest tree in the forest T created by deleting that node from T.

For example, consider the tree:



Deleting node 4 yields two trees whose member nodes are {5} and {1,2,3,6,7}. The larger of these two trees has five nodes, thus the balance of node 4 is five. Deleting node 1 yields a forest of three trees of equal size: {2,6}, {3,7}, and {4,5}. Each of these trees has two nodes, so the balance of node 1 is two.

For each input tree, calculate the node that has the minimum balance. If multiple nodes have equal balance, output the one with the lowest number.

The input trees are given one per file. Input cases are numbered 1 through 10 and are available via the contest website.

Input:

- The first line of input contains a single integer K, specifying the input case number.
- The second line contains a single integer N.
- Each of the next N-1 lines contains two space-separated node numbers that are the endpoints of an edge in the tree. No edge will be listed twice, and all edges will be listed.

Example input:

```
0
7
2 6
1 2
1 4
4 5
3 7
3 1
```

Output:

- The first line of output should contain the string "FILE balance K", where K is the input case number.
- The second line should contain a single integer, the number of the node with minimum balance.
- The third line should contain a single integer, the balance of that node.

Example output:

```
FILE balance 0
1
2
```

CONSTRAINTS:

Running time	1 second of CPU
Memory	64 MB

SCORING

You will receive full points on each test case for which your program produces a correct output file. No partial credit will be given on any test case.

Guess Which Cow (interactive task)

TASK

The N ($1 \le N \le 50$) cows in Farmer John's herd look very much alike and are numbered 1…N. When Farmer John puts a cow to bed in her stall, he must determine which cow he is putting to bed so he can put her in the correct stall.

Cows are distinguished using P ($1 \le P \le 8$) properties, numbered 1…P, each of which has three possible values. For example, the color of a cow's ear tag might be yellow, green, or red. For simplicity, the values of every property are represented as the letters '`X`', '`Y`', and '`Z`'. Any pair of Farmer John's cows will differ in at least one property.

Write a program that, given the properties of the cows in Farmer John's herd, helps Farmer John determine which cow he is putting to bed. Your program can ask Farmer John no more than 100 questions of the form: Is the cow's value for some property T in some set S? Try to ask as few questions as possible to determine the cow.

Input: `guess.in`

- The first line of the input file contains two space-separated integers, N and P.
- Each of the next N lines describes a cow's properties using P space-separated letters. The first letter on each line is the value of property 1, and so on. The second line in the input file describes cow 1, the third line describes cow 2, etc.

Example input:

```
4 2
X Z
X Y
Y X
Y Y
```

Interactivity: *standard input and output*

The question/answer phase takes place via standard input and standard output.

Your program asks a question about the cow being put to bed by writing to standard output a line that is a '`Q`' followed by a space, the property number, a space, and a space-separated set of one or more values. For example, "`Q 1 Z Y`" means "Does property 1 have value either '`Z`' or '`Y`' for the cow being put to bed?". The property must be an integer in the range 1…P. All values must be '`X`', '`Y`', or '`Z`' and no value should be listed more than once for a single question.

After asking each question your program asks, read a single line containing a single integer. The integer 1 means the value of the specified property of the cow being put to bed is in the set of values given; the integer 0 means it is not.

The program's last line of output should be a '`C`' followed by a space and a single integer that specifies the cow that your program has determined Farmer John is putting to bed.

Example exchange (for example input above):

Input	*Output*	*Explanation*
	`Q 1 X Z`	
`0`		Could be cow 3 or cow 4.
	`Q 2 Y`	
`1`		Must be cow 4!
	`C 4`	
program exits		

CONSTRAINTS

Running time	1 second of CPU
Memory	64 MB

SCORING

Correctness: 30% of points

Programs will receive full score on correctness only if the cow specified is the only cow that is consistent with the answers given. A program that asks more than 100 questions for a test case will receive no points for that test case.

Question count: 70% of points

The remaining points will be determined by the number of questions required to correctly determine the cow. The test cases are designed to reward minimizing the worst-case question count. Partial credit will be given for near-optimal question counts.

Test Data for Guess Which Cow

Algorithms:

Algorithm 1: Perform a breadth-first search on the question set, where each state is the set of the remaining possible cows.

While this state space sounds like there are 2^{50} possible states, this is not true. The state can also be viewed as the set of possible values for each attribute. There are $2^3 - 1$ values for the set of possible values for each attribute, for a total of less than 2^{24} total states (it is more like $2^{22.5}$ states).

This algorithm runs in $O(N \cdot P \cdot 2^3 \cdot 2^{3P})$ time. It is expected to receive full marks, if implemented well, which may require the use of bit vectors instead of arrays of Boolean values.

Algorithm 2: At each step, select the question which splits the cows most evenly.

This algorithm runs in $O(N \cdot P \cdot 2^3)$ time. It is expected to get around 85% of the points.

Algorithm 3: Ask two questions about each property to determine the exact value, stopping when the cow is known, as well as if you know the property value after the first question about that property.

This algorithm runs very quickly. It is expected to get around 70% of the points.

Algorithm 4: Ask two questions about each property to determine the exact value.

This algorithm runs very quickly. It is expected to get around 45% of the points.

Test Data

Test #	Points	N	P	Min Q's
1	5	10	3	4
2	5	6	4	3
3	5	7	4	3
4	5	8	4	3
5	5	10	4	4
6	5	15	4	4
7	5	16	4	4
8	5	27	3	6
9	5	30	5	6
10	5	40	6	6
11	5	50	7	7
12	5	50	8	7
13	5	50	8	7
14	5	50	4	6
15	5	50	5	6
16	5	50	8	6
17	5	50	4	7
18	5	50	8	6
19	5	17	8	9
20	5	45	8	10

Scoring Methodology:

Optimal question count	100%
1 extra question	80%
2 extra questions	50%
3 to 5 extra questions	40%
More than 5 extra questions	30%

Amazing Robots

TASK

You are the proud owner of two robots that are located in separate rectangular mazes. Square (1, 1) in a maze is the square in the upper-left corner, which is the north-west corner. Maze i (i = 1, 2) has a set of G_i ($0 \le G_i \le 10$) guards trying to capture the robots by patrolling back and forth on a straight patrol path. Your goal is to determine a sequence of commands such that the robots exit the mazes without either robot being captured by a guard.

At the beginning of each minute, you broadcast the same command to both robots. Each command is a direction (north, south, east, or west). A robot moves one square in the direction of the command, unless the robot would collide with a wall, in which case the robot does not move for that minute. A robot exits the maze by walking out of it. A robot ignores commands after it has exited its maze.

Guards move one square at the beginning of each minute, at the same time as the robots. Each guard begins at a given square facing a given direction and moves forward one square per minute until the guard has moved one fewer square than the number of squares in his patrol path. The guard then turns around instantaneously and walks in the opposite direction back to his starting square, where he turns around again and repeats his patrol path until each robot has exited its maze.

A guard's patrol will not require the guard to walk through walls or exit the maze. Although guard patrols may overlap, no two guards will ever collide: they will never occupy the same square at the end of a minute, and they will not exchange squares with each other during a minute. A guard in a maze will not start in the same square as the robot in that maze.

A guard captures a robot if the guard occupies the same square at the end of a minute as the robot, or if the guard and the robot exchange squares with each other during a minute.

Given two mazes (each no larger than 20x20) along with the initial square of each robot and the patrol paths of the guards in each maze, determine a sequence of commands for which the robots exit without being caught by the guards. Minimize the time it takes for the later robot to exit its maze. If the robots exit at different times, the time at which the earlier robot exited does not matter.

Input: `robots.in`

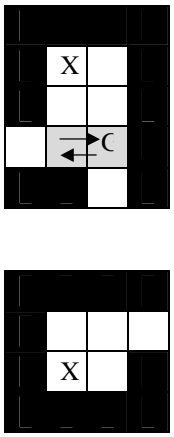
The first set of lines describes the first maze and its occupants. Subsequently, the second set of lines describes the second maze and its occupants.

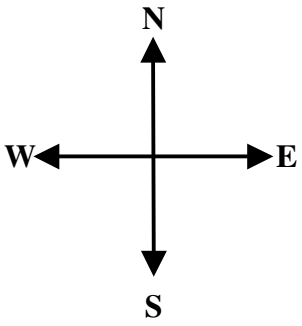
- The first line of input contains two space-separated integers, R_1 and C_1, the number of rows and columns in maze 1.
- The next R_1 lines each contain C_1 characters specifying the maze layout. The robot's starting square is specified by an '`X`'. A '`.`' represents an open space and '#' represents a wall. Each maze will contain exactly one robot.
- Following the maze layout is a single line with a single integer G_1, the number of guards in the first maze ($0 \le G_1 \le 10$).
- Each of the next G_1 lines describes a guard's patrol as three integers and a character, separated by single spaces. The first two integers specify the row and column of the starting square of the guard. The third integer specifies the number of squares (2…4) in the guard's patrol path. The character specifies the initial direction the guard is facing: '`N`', '`S`', '`E`', or '`W`' (north, south, east, and west, respectively).

The description of the second maze follows the description of the first, in the same format but with potentially different values.

Example input:

```
5 4
####
#X.#
#..#
...#
##.#
1
4 3 2 W
4 4
####
#...
#X.#
####
0
```





Output: `robots.out`

The first line of the output should contain a single integer K ($K \le 10000$), the number of commands for both robots to exit the maze without being captured. If such a sequence of commands exists, the shortest sequence will have no more than 10000 commands. The next K lines are the sequence of commands, each containing a single character from the set {'`N`', '`S`', '`E`', '`W`'}. If no such sequence exists, output a single line containing "`-1`".

Both robots should exit their mazes by the end of the commands. The last command should cause at least one of the robots to exit its maze. If multiple sequences of commands cause the robots to exit in the minimum time, any will be accepted.

Example output:

```
8
E
N
E
S
S
S
E
S
```

CONSTRAINTS

Running time	2 seconds of CPU
Memory	64 MB

SCORING

No partial credit will be given on test cases for which no sequence of commands exists. Partial credit for other test cases will be given as described below.

Correctness: 20% of points

The output file for a test case is considered correct if it is correctly formatted, contains no more than 10000 commands, and the sequence of commands causes the robots to exit the mazes, with the last command causing at least one robot to exit its maze.

Minimality: 80% of points

The output file for a test case is considered minimal if it is correct and there is no shorter sequence of commands that is correct. A program whose sequence of commands is not minimal will receive no points for minimality.

Test Data for Amazing Robots

Algorithms:

Note that the guards simultaneously return to their starting location every 12 minutes.

For analysis purposes, assume that the mazes are of equal size. More over, let *n* be the number of squares in the maze (R·C).

Algorithm 1: Perform a breadth-first search where the state of the search is the location of the two robots and the current minute within the guard cycle of 12 minutes. Use memorization to ensure that no state is visited more than once.

This algorithm takes $O(12\ n^2)$ time. It is expected to receive full marks, although this may vary based on implementation.

Algorithm 2: Perform breadth-first search where the state of the search is the location of the two robots and the current minute. Use memorization (with a hash table) to ensure that no state is visited more than once.

This algorithm takes $O(12\ T\ n^2)$ time, where T is the minimum amount of time it takes for both robots to exit their maze. It is expected to receive about 65% of the points.

Algorithm 3: Perform breadth-first search where the state is the location of the two robots and the time. Do not eliminate duplications.

This algorithm takes $O(4^T)$ time. It is expected to receive about 40% of the points.

Algorithm 4: Perform breadth-first search where the state is the location of the two robots and the value of the state is minimum time it takes to obtain that state. Do not visit any state more than once.

This algorithm takes $O(n^2)$ time, but is not always correct. It only considers the shortest path to each location pair. It may be of value to get to a location pair later, as you cannot waste time in the middle of an open space.

This algorithm is expected to receive about 30% of the points.

The efficiency of this algorithm varies based on the outer algorithm used. This algorithm is expected to get about 45% of the points, although this will vary depending on algorithm and implementation.

Test Data

Test #	Points	Maze 1	Maze 2	Guards	Answer
1	5	1x1	5x4	0	6
2	5	5x4	5x4	2	10
3	5	5x5	3x3	2	9
4	5	8x8	8x8	5	9
5	5	8x8	8x8	6	17
6	5	5x8	8x8	5	8
7	5	8x8	8x8	7	13
8	5	8x8	8x8	8	15
9	5	4x8	7x8	4	15
10	5	12x12	13x13	5	49
11	5	12x12	13x13	14	-1
12	5	12x12	13x13	13	41
13	5	20x20	20x20	3	66
14	5	20x20	15x20	4	68
15	5	20x20	20x20	5	200
16	5	20x20	20x20	19	200
17	5	20x20	20x20	20	-1
18	5	20x20	20x20	20	99
19	5	20x20	20x20	20	314
20	5	20x20	20x20	20	328

Seeing the Boundary

TASK

Farmer Don watches the fence that surrounds his N meter by N meter square, flat field ($2 \le N \le 500{,}000$). One fence corner is at the origin (0, 0) and the opposite corner is at (N, N); the sides of Farmer Don's fence are parallel to the X and Y axes.

Fence posts appear at all four corners and also at every meter along each side of the fence, for a total of $4 \cdot N$ fence posts. The fence posts are vertical and are considered to have no radius. Farmer Don wants to determine how many of his fence posts he can watch when he stands at a given location within his fence.

Farmer Don's field contains R ($1 \le R \le 30{,}000$) huge rocks that obscure his view of some fence posts, as he is not tall enough to look over any of these rocks. The base of each rock is a convex polygon with nonzero area whose vertices are at integer coordinates. The rocks stand completely vertical. Rocks do not overlap, do not touch other rocks, and do not touch Farmer Don or the fence. Farmer Don does not touch the fence, does not stand within a rock, and does not stand on a rock.

Given the size of Farmer Don's fence, the locations and shapes of the rocks within it, and the location where Farmer Don stands, compute the number of fence posts that Farmer Don can see. If a vertex of a rock lines up perfectly with a fence post from Farmer Don's location, he is not able to see that fence post.

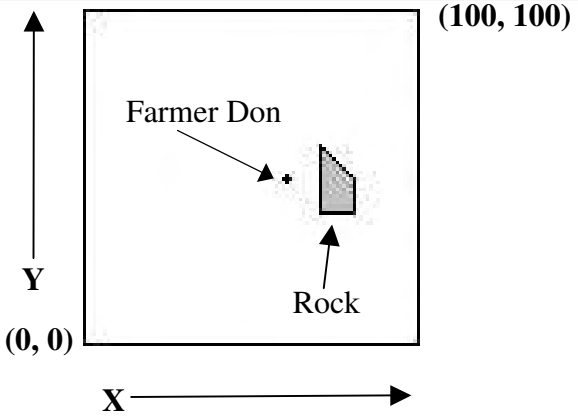
Input: `boundary.in`

- The first line of input contains two space-separated integers: N and R.
- The next line of input contains two space-separated integers that specify the X and Y coordinates of Farmer Don's location inside the fence.
- The rest of the input file describes the R rocks:
  - Rock i's description starts with a line containing a single integer p_i ($3 \le p_i \le 20$), the number of vertices in the rock's base.
  - Each of the next p_i lines contains a space-separated pair of integers that are the X and Y coordinates of a vertex. The vertices of a rock's base are distinct and given in counterclockwise order.

Example input:

```
100 1
60 50
5
70 40
75 40
80 40
80 50
70 60
```

Note that the base of rock 1 has three collinear vertices: (70,40), (75,40), and (80,40)



Output: `boundary.out`

The output file should contain a single line with a single integer, the number of fence posts visible to Farmer Don.

Example output:

```
319
```

CONSTRAINTS

Running time	1 second of CPU
Memory	64 MB

SCORING

You will receive full points on each test case for which your program produces a correct output file. No partial credit will be given on any test case.

Test Data for Seeing the Boundary

Algorithms:

Note that there are many ways to get the computational geometry wrong. In addition, one must be very careful to avoid rounding mistakes in floating point numbers. Our low scoring solutions make errors of various sorts, and are not covered below.

Let d be the maximum number of vertices in any rock.

Algorithm 1: Consider the fence as a circular list. Each rock obscures a range of fence posts. Determine the range for each fence post by sorting the angles of the rocks from Farmer Don's location and find the right-most and left-most rock obscured (being careful to deal with wrap-around). Collect these ranges. After processing all rocks, sort the ranges and determine loop through the ranges, counting any fence post not covered by a range.

This algorithm takes O(R log R + N log d) time. It is expected to receive full marks

Algorithm 2: Keep a binary array of fence posts. For each rock, determine the range obscured by the rock and reset those data points.

This algorithm takes O(N R) time. It is expected to receive about 80% of the points.

Algorithm 3: For each rock and fence post, run a ray to determine if the rock obscures the fence posts.

This algorithm takes O(N R) time. It is expected to receive about 72% of the points.

Test Data

Test #	Points	N	R	Answer
1	4	100	1	220
2	4	100	1	220
3	4	100	1	218
4	4	100	1	246
5	4	100	1	204
6	4	100	1	269
7	4	40	1	144
8	4	40	1	144
9	4	1500	100	2257
10	4	4000	1500	13
11	4	5000	1800	27
12	4	6000	2000	38
13	4	8000	2400	8244
14	4	10000	3000	4
15	4	16000	3700	0
16	4	28000	5000	108
17	4	40000	6400	288
18	4	30000	373	88507
19	4	62000	9000	83
20	4	91000	12000	368
21	4	123000	15000	43
22	4	164000	17000	20
23	4	250000	23000	2663
24	4	340000	23000	28
25	4	500000	30000	2518

August 22, 2003

Contestant Score List

				Day One			Day Two			
Rank	**Medal**	**Contestant**		**M**	**C**	**R**	**G**	**R**	**B**	**Total**
1	Gold	Hwan-Seung Yeo	KOR	100	50	49.4	60.0	100	96	455.4
2	Gold	Ivaylo Riskov	BGR	100	85	49.4	85.5	8	100	427.9
3	Gold	Thomas Wuerthinger	AUT	100	75	49.4	85.5	80	36	425.9
4	Gold	Huy Nguyen Le	VNM	100	30	48.8	85.5	60	100	424.3
5	Gold	Victor-Marius Costan	ROM	100	100	12.0	90.0	90	32	424.0
6	Gold	Alex Schwendner	USA	100	100	49.4	95.0	70	8	422.4
7	Gold	Pablo Dal Lago	ARG	100	35	40.4	77.5	65	100	417.9
8	Gold	Erik Bernhardsson	SWE	100	60	38.0	97.0	90	32	417.0
9	Gold	Tiankai Liu	USA	100	5	64.4	70.0	100	72	411.4
10	Gold	Bartosz Walczak	POL	100	75	87.6	42.0	100	0	404.6
11	Gold	Martin Orr	IRL	65	45	44.6	85.5	100	64	404.1
12	Gold	Dmitry Orlov	RUS	100	45	48.8	40.0	60	96	389.8
*		Eric Price	USB	100	0	63.4	86.5	65	72	386.9
13	Gold	Won-Sik Kim	KOR	100	50	48.8	66.0	90	24	378.8
14	Gold	Milan Straka	CZE	100	15	12.0	83.5	85	80	375.5
15	Gold	Chi Man Liu	HKG	100	55	12.0	84.0	100	24	375.0
16	Gold	Veli Peltola	FIN	100	5	87.6	85.5	50	44	372.1
17	Gold	Lin He	CHN	100	30	49.4	100.0	10	68	357.4
18	Gold	Bogdan Yakovenko	UKR	100	35	49.4	72.5	100	0	356.9
19	Gold	Luka Kalinovcic	HRV	100	70	51.3	85.5	30	20	356.8
20	Gold	Kuen-Bang Hou	TPE	100	25	49.4	85.5	0	96	355.9
21	Gold	Radu Berinde	ROM	100	30	49.4	67.5	35	72	353.9
22	Gold	Jakub Zavodny	SVK	100	20	49.4	85.5	10	88	352.9
23	Gold	Carl Nettelblad	SWE	90	100	43.4	84.0	35	0	352.4
24	Gold	Filip Wolski	POL	100	30	49.4	85.5	10	76	350.9
25	Silver	Tai-Hsu Lin	TPE	100	25	49.4	85.5	0	84	343.9
26	Silver	Zhilei Xu	CHN	100	65	49.4	4.0	25	100	343.4
27	Silver	Gábor Pelládi	HUN	100	25	63.2	85.5	65	4	342.7
28	Silver	Ivan Anev	BGR	100	75	49.4	85.5	0	32	341.9
29	Silver	Paul Jefferys	GBR	75	75	49.4	85.5	10	40	334.9
30	Silver	Nattavut Yampikulsakul	THA	85	20	49.4	85.5	10	84	333.9
31	Silver	Ville Mäkynen	FIN	90	20	48.8	85.5	10	76	330.3
32	Silver	Chan-Il Jung	KOR	100	0	49.4	96.0	10	72	327.4
33	Silver	Young-Sub Bae	KOR	100	0	49.4	85.5	15	76	325.9
33	Silver	Ronald Van Der Velden	NLD	75	30	49.4	85.5	50	36	325.9
33	Silver	Nawanol Theera-Ampornpunt	THA	100	5	49.4	85.5	10	76	325.9
*		Adam Rosenfield	USB	60	20	29.4	85.5	35	96	325.9
36	Silver	Semen Dyatlov	RUS	100	15	35.4	70.0	10	92	322.4
37	Silver	Parinda Jayasiri	LKA	85	50	49.4	85.5	15	36	320.9
38	Silver	Juha Arpiainen	FIN	100	25	87.6	66.5	10	28	317.1

August 22, 2003

Contestant Score List

				Day One			Day Two			
Rank	**Medal**	**Contestant**		**M**	**C**	**R**	**G**	**R**	**B**	**Total**
39	Silver	Jonathan Mosheiff	ISR	85	25	20.4	85.5	0	100	315.9
40	Silver	Uladzimir Kerus	BLR	75	10	48.8	85.5	10	84	313.3
41	Silver	Aleksandar Ilic	SEM	65	0	49.4	85.5	10	100	309.9
42	Silver	Matei Zaharia	CAN	100	50	42.8	90.0	25	0	307.8
43	Silver	Pavel Cizek	CZE	100	60	47.6	85.5	5	8	306.1
44	Silver	Tung Cao Thanh	VNM	65	30	47.0	85.5	5	72	304.5
45	Silver	Xide Lin	CHN	100	5	25.2	90.0	55	28	303.2
46	Silver	Anders Kaseorg	USA	65	25	49.4	68.0	5	88	300.4
47	Silver	Marcin Michalski	POL	100	5	12.0	88.5	90	4	299.5
48	Silver	Timothy Abbott	USA	10	30	44.6	85.5	100	28	298.1
49	Silver	Heinrich-Gregor Zirnstein	DEU	100	0	15.0	85.5	10	80	290.5
50	Silver	Stefan Ciobaca	ROM	85	30	49.4	85.5	40	0	289.9
51	Silver	Michal Burger	SVK	100	5	72.0	72.0	10	28	287.0
52	Silver	Karlis Gangis	LVT	90	40	50.6	85.5	10	8	284.1
53	Silver	Simon Parent	CAN	60	40	49.4	86.5	48	0	283.9
54	Silver	Mark Thompson	GBR	100	15	50.0	86.5	0	28	279.5
55	Silver	Giuseppe Ottaviano	ITA	100	55	40.4	82.5	0	0	277.9
56	Silver	Davor Bonaci	HRV	100	30	12.0	85.5	34	16	277.5
57	Silver	Mihkel Kree	EST	100	0	49.4	26.0	0	100	275.4
58	Silver	Sören Petersen	SWE	100	20	49.4	85.5	15	0	269.9
59	Silver	Tofig Hasanov	AZE	90	5	44.6	77.5	10	40	267.1
60	Silver	Linsen Loots	ZAF	85	25	49.4	85.5	10	12	266.9
61	Silver	Dan Ghinea	ROM	100	10	49.4	56.0	51	0	266.4
62	Silver	Gilberto Abram	ITA	100	30	49.4	85.5	0	0	264.9
62	Silver	Vilius Naudziunas	LTU	100	0	43.4	77.5	0	44	264.9
64	Silver	Alexander Hullmann	DEU	100	15	49.4	90.0	0	8	262.4
65	Silver	Joel Jonsson	SWE	25	0	48.8	77.5	10	100	261.3
66	Silver	Abdullah Akce	TUR	100	25	45.8	85.5	0	4	260.3
67	Silver	Prachaya Phaisanwiphatpong	THA	100	10	43.4	82.5	0	24	259.9
68	Silver	Luka Dondjivic	HRV	85	30	45.2	85.5	10	4	259.7
69	Silver	Lucien Pech	FRA	70	0	42.2	85.5	0	60	257.7
70	Bronze	Hakan Yildiz	TUR	100	0	49.4	82.5	21	0	252.9
*		Boris Alexeev	USB	30	10	49.4	85.5	70	8	252.9
71	Bronze	Duc Pham Tran	VNM	100	0	49.4	85.5	15	0	249.9
72	Bronze	Qiming Hou	CHN	100	100	49.4	0.0	0	0	249.4
73	Bronze	Sander Land	NLD	80	40	49.4	77.5	0	0	246.9
74	Bronze	Sidney Hok Nang Fong	HKG	100	0	45.8	85.5	10	4	245.3
75	Bronze	Thang Dinh Ngoc	VNM	100	0	41.6	86.5	0	16	244.1
76	Bronze	Bernhard Walzer	AUT	50	5	42.2	85.5	55	4	241.7
77	Bronze	Veselin Georgiev	BGR	90	20	12.0	76.0	15	28	241.0

				Day One			Day Two			
Rank	**Medal**	**Contestant**		**M**	**C**	**R**	**G**	**R**	**B**	**Total**
78	Bronze	Nicholas Jimsheleishvili	GEO	100	5	49.4	74.0	10	0	238.4
79	Bronze	Mattias Linnap	EST	100	0	51.2	85.5	0	0	236.7
80	Bronze	Oleksandr Galkin	UKR	50	15	48.8	85.5	5	32	236.3
81	Bronze	Frantisek Simancik	SVK	80	0	46.4	85.5	0	24	235.9
82	Bronze	Matteo Bruni	ITA	90	5	39.8	85.5	15	0	235.3
83	Bronze	Michal Brzozowski	POL	100	0	42.0	83.0	10	0	235.0
84	Bronze	Ivo List	SVN	50	35	49.4	85.5	10	4	233.9
85	Bronze	Chethiya Abeysinghe	LKA	85	5	50.6	90.0	0	0	230.6
86	Bronze	Svetoslav Kolev	BGR	65	0	6.0	60.5	10	88	229.5
87	Bronze	Shen Tian	ZAF	85	5	49.4	85.5	0	4	228.9
88	Bronze	Ishan Behoora	IND	25	60	49.4	89.0	5	0	228.4
89	Bronze	Liang Junjie	SGP	25	25	49.4	85.5	10	32	226.9
90	Bronze	Maksim Osipau	BLR	100	0	31.4	42.0	53	0	226.4
91	Bronze	Zviad Metreveli	GEO	90	0	12.0	85.5	10	28	225.5
92	Bronze	Marijn Kruisselbrink	NLD	75	5	49.4	77.5	10	8	224.9
93	Bronze	Gintautas Miliauskas	LTU	75	5	49.4	78.5	0	16	223.9
94	Bronze	Guilherme Issao C. Fujiwara	BRA	100	0	24.0	85.5	0	12	221.5
95	Bronze	Giancarlo Salamanca	AUS	20	5	46.4	77.5	10	60	218.9
96	Bronze	Matej Jan	SVN	75	30	49.4	0.0	10	52	216.4
97	Bronze	David Barr	AUS	100	5	25.4	85.5	0	0	215.9
98	Bronze	Evgeny Shavlugin	RUS	25	15	46.4	90.0	5	32	213.4
99	Bronze	Patrick Coleman	AUS	25	25	47.6	85.5	10	20	213.1
100	Bronze	Béla Rácz	HUN	100	0	87.6	0.0	1	24	212.6
101	Bronze	Ignat Meldin	RUS	5	20	49.4	75.0	10	52	211.4
102	Bronze	Markus Ojala	FIN	100	5	49.4	26.0	10	20	210.4
103	Bronze	Arturs Verza	LVT	25	25	42.0	90.0	0	28	210.0
104	Bronze	Anders Rudebeck	DNK	100	0	29.4	77.5	0	0	206.9
105	Bronze	Pavel Sakau	BLR	40	25	49.4	81.0	10	0	205.4
106	Bronze	Aleksandar Zlateski	SEM	45	5	49.4	85.5	0	20	204.9
106	Bronze	Jurij Kodre	SVN	75	0	28.4	85.5	0	16	204.9
108	Bronze	Cheng-Yang Tsai	TPE	35	10	49.4	85.5	0	24	203.9
109	Bronze	Olegs Urziks	LVT	100	5	19.8	77.5	0	0	202.3
110	Bronze	Vahe Musoyan	ARM	45	15	42.8	86.5	10	0	199.3
111	Bronze	Tomas Gavenciak	CZE	100	0	0.0	89.0	6	0	195.0
*		Brian Jacokes	USB	0	70	12.0	84.5	0	28	194.5
112	Bronze	Alejandro Martin Deymonnaz	ARG	30	25	49.4	85.5	0	4	193.9
113	Bronze	Swarnendu Datta	IND	35	5	49.4	85.5	10	8	192.9
114	Bronze	Igor Kabiljo	SEM	0	20	54.2	85.5	0	32	191.7
115	Bronze	Jucovschi Constantin	MDA	100	30	18.0	7.5	0	36	191.5
116	Bronze	Aasmund Eldhuset	NOR	100	0	12.0	78.5	0	0	190.5

August 22, 2003

Contestant Score List

				Day One			Day Two			
Rank	**Medal**	**Contestant**		**M**	**C**	**R**	**G**	**R**	**B**	**Total**
117	Bronze	Shek Ming Sherman Lam	HKG	0	25	49.4	85.5	0	24	183.9
118	Bronze	Pichayut Jirapinyo	THA	25	30	36.8	81.5	10	0	183.3
119	Bronze	Yuriy Znovyak	UKR	25	5	48.8	85.5	10	8	182.3
120	Bronze	Andrew Doumanoglou	GRC	100	0	24.0	0.0	10	48	182.0
121	Bronze	Emir Kumalic	BIH	70	0	36.6	42.0	0	32	180.6
122	Bronze	Sorbalo Eugen	MDA	100	0	25.8	0.0	10	44	179.8
123	Bronze	David Steurer	DEU	100	0	49.4	0.0	30	0	179.4
124	Bronze	Kwong Fai Tsang	HKG	85	5	45.8	33.0	10	0	178.8
125	Bronze	Simon Felix	CHE	25	15	47.6	85.5	5	0	178.1
126	Bronze	Marko Zivkovic	HRV	0	0	50.4	85.5	0	40	175.9
127	Bronze	Nikola Postolov	MKD	70	5	12.0	77.5	10	0	174.5
128	Bronze	Sarge Rogatch	BLR	0	30	48.8	85.5	10	0	174.3
128	Bronze	Anton Nikolayev	KAZ	0	35	48.8	85.5	5	0	174.3
130	Bronze	Indraneel Mukherjee	IND	100	0	0.0	50.0	0	24	174.0
131	Bronze	Benoît Roche	FRA	70	0	29.4	34.0	0	40	173.4
132	Bronze	Sargis Gevorgyan	ARM	85	0	24.6	63.5	0	0	173.1
133				25	35	12.0	66.0	10	24	172.0
133				85	0	47.0	12.0	0	28	172.0
135				25	5	47.6	84.0	10	0	171.6
136				30	5	49.4	53.0	10	24	171.4
137				25	20	25.8	90.0	10	0	170.8
138				35	20	41.0	55.0	0	16	167.0
139				100	5	49.4	11.0	0	0	165.4
140				65	0	30.0	42.0	0	28	165.0
141				80	0	32.6	48.0	0	4	164.6
142				25	0	41.6	87.5	10	0	164.1
143				0	5	49.4	85.5	0	24	163.9
143				25	0	49.4	85.5	0	4	163.9
145				0	20	49.4	85.5	0	8	162.9
146				90	0	12.0	14.5	10	36	162.5
147				100	5	36.8	5.0	10	4	160.8
148				50	25	12.0	48.5	25	0	160.5
148				30	0	24.0	85.5	5	16	160.5
150				25	0	49.4	85.5	0	0	159.9
151				30	0	27.0	85.5	0	16	158.5
152				25	25	36.0	58.0	10	4	158.0
153				100	0	49.4	8.0	0	0	157.4
154				45	25	39.2	22.0	10	16	157.2
155				25	25	48.8	53.0	0	4	155.8
156				0	30	49.4	75.5	0	0	154.9

August 22, 2003

Contestant Score List

			Day One			Day Two			
Rank	**Medal**	**Contestant**	**M**	**C**	**R**	**G**	**R**	**B**	**Total**
157			30	0	39.8	74.5	10	0	154.3
158			0	0	49.4	90.0	10	4	153.4
159			25	5	27.0	84.0	0	8	149.0
160			40	5	38.6	65.0	0	0	148.6
161			5	5	42.0	85.5	10	0	147.5
162			80	15	49.4	3.0	0	0	147.4
163			30	5	24.6	85.5	0	0	145.1
164			25	25	49.4	41.5	0	4	144.9
165			25	5	24.0	48.5	10	32	144.5
165			25	0	24.0	85.5	10	0	144.5
167			0	25	48.2	48.5	10	12	143.7
168			0	10	49.4	77.5	0	4	140.9
169			0	5	49.4	85.5	0	0	139.9
170			0	5	34.8	85.5	10	0	135.3
171			0	5	24.0	85.5	15	4	133.5
172			35	10	12.0	72.0	0	4	133.0
173			25	5	49.4	31.5	0	20	130.9
174			50	0	32.6	8.0	10	28	128.6
175			25	0	12.0	84.0	0	4	125.0
176			40	0	33.6	40.0	10	0	123.6
177			25	0	12.0	86.5	0	0	123.5
178			85	0	35.4	0.0	0	0	120.4
179			20	5	45.8	48.5	0	0	119.3
180			0	5	24.0	85.5	0	4	118.5
181			25	15	4.8	72.5	0	0	117.3
182			25	30	4.8	56.0	0	0	115.8
183			0	0	29.4	85.5	0	0	114.9
184			0	0	27.6	85.5	0	0	113.1
185			0	5	22.8	57.5	5	20	110.3
186			0	0	43.4	65.0	0	0	108.4
187			20	15	24.0	48.5	0	0	107.5
187			25	5	12.0	65.5	0	0	107.5
189			0	0	23.4	79.5	0	4	106.9
190			20	5	24.6	56.5	0	0	106.1
191			0	0	12.0	92.5	0	0	104.5
192			50	5	49.4	0.0	0	0	104.4
193			0	0	14.4	85.5	0	0	99.9
194			0	5	36.0	57.5	0	0	98.5
195			55	0	18.0	0.0	0	24	97.0
196			20	15	49.4	0.0	10	0	94.4

			Day One			Day Two			
Rank	**Medal**	**Contestant**	**M**	**C**	**R**	**G**	**R**	**B**	**Total**
197			0	0	27.0	67.0	0	0	94.0
198			25	0	25.8	15.0	0	28	93.8
199			25	0	24.0	42.0	0	0	91.0
200			25	0	49.4	0.0	16	0	90.4
201			25	0	17.4	45.5	0	0	87.9
202			0	5	12.0	70.5	0	0	87.5
203			0	0	33.6	50.0	0	0	83.6
204			25	5	49.4	0.0	0	4	83.4
205			0	0	24.0	57.5	0	0	81.5
206			25	0	12.0	43.0	0	0	80.0
207			30	0	42.2	0.0	0	0	72.2
208			20	0	12.0	38.0	0	0	70.0
209			15	20	10.8	6.0	10	8	69.8
210			0	0	49.4	0.0	0	20	69.4
211			0	0	12.0	57.0	0	0	69.0
212			0	15	48.8	0.0	0	0	63.8
213			0	0	24.0	24.5	10	4	62.5
213			0	10	6.0	46.5	0	0	62.5
215			0	5	12.0	44.5	0	0	61.5
216			40	0	6.0	0.0	10	4	60.0
217			30	0	29.6	0.0	0	0	59.6
218			30	5	24.0	0.0	0	0	59.0
219			0	25	27.0	0.0	0	4	56.0
220			25	0	20.4	0.0	10	0	55.4
221			0	0	12.0	38.5	0	0	50.5
222			20	15	15.0	0.0	0	0	50.0
223			0	5	44.8	0.0	0	0	49.8
224			0	5	0.0	34.5	10	0	49.5
225			0	0	38.6	10.5	0	0	49.1
226			0	5	42.0	0.0	0	0	47.0
227			40	0	0.0	0.0	0	4	44.0
228			30	0	12.6	0.0	0	0	42.6
229			0	25	12.0	0.0	0	4	41.0
230			0	0	22.2	8.0	10	0	40.2
231			0	5	24.0	0.0	10	0	39.0
232			0	0	34.2	0.0	0	4	38.2
233			0	0	36.8	0.0	0	0	36.8
234			0	5	27.6	0.0	0	4	36.6
235			0	5	24.6	5.0	0	0	34.6
236			0	5	12.0	0.0	10	4	31.0

August 22, 2003

Contestant Score List

			Day One			Day Two			
Rank	**Medal**	**Contestant**	**M**	**C**	**R**	**G**	**R**	**B**	**Total**
237			0	0	12.0	17.0	0	0	29.0
238			0	0	12.0	0.0	0	16	28.0
239			0	0	17.4	0.0	10	0	27.4
240			20	0	6.0	0.0	0	0	26.0
241			0	0	15.0	0.0	10	0	25.0
242			0	5	17.4	0.0	0	0	22.4
243			0	5	15.6	0.0	0	0	20.6
244			0	0	0.0	20.0	0	0	20.0
245			0	5	12.0	0.0	1	0	18.0
245			0	0	18.0	0.0	0	0	18.0
247			0	0	17.4	0.0	0	0	17.4
248			0	5	0.0	0.0	10	0	15.0
249			0	0	0.0	4.0	10	0	14.0
249			0	0	12.0	2.0	0	0	14.0
251			0	0	12.6	0.0	0	0	12.6
252			0	0	12.0	0.0	0	0	12.0
252			0	0	12.0	0.0	0	0	12.0
252			0	0	12.0	0.0	0	0	12.0
255			0	0	4.8	6.5	0	0	11.3
256			0	0	0.0	0.0	10	0	10.0
257			0	5	0.0	0.0	0	0	5.0
258			0	0	0.0	0.0	0	4	4.0
259			0	0	0.0	0.0	0	0	0.0
259			0	0	0.0	0.0	0	0	0.0
259			0	0	0.0	0.0	0	0	0.0
259			0	0	0.0	0.0	0	0	0.0
259			0	0	0.0	0.0	0	0	0.0
259			0	0	0.0	0.0	0	0	0.0
259			0	0	0.0	0.0	0	0	0.0