2015 UTS Programming Competition

Saturday 14 March, 2015

Painstakingly prepared by the UTS Programmers' Society (ProgSoc)

Sponsored by
The School of Computing and Communications, Faculty of Engineering & IT and WiseTech Global

*Contains the **competition question set** and an **Appendix** following the questions.*

`www.progsoc.org`

Problem 1: ASCII Chequerboards

Run time limit: 2 seconds

Problem Description

Let's kick off our competition with some fun and games!

If you have ever played board games such as chess or chequers, or at least watched these games being played, you will have undoubtedly encountered the game board upon which these games are played. You would have noticed the distinct, alternating two-toned pattern of smaller squares arranged within a larger square that is the game board, upon which the game pieces were placed, moved about and removed from.

Your task is to write a program that recreates that pattern. In lieu of black and white squares, you will instead use characters from the keyboard -- an ASCII chequerboard!

"This is a pointless task!", you may well say. "*Au contraire!*", is my rejoinder, for in order to succeed in this competition, you will need to be sure that you know how to correctly read input, write output, and follow specifications. Consider this task to be a sanity test of sorts.

Data Specification

Input

A single integer, `N`, where 2 <= `N` <= 40, denoting the size of your chequerboard.

Output

There will be `N` horizontal lines in total. For clarity, we shall refer to the first line as line `1` (which is an "odd-numbered line") and the last line as line `N` (which can be odd or even).

The first character to print on odd-numbered lines will be an asterisk (*). The following character will be an underscore (_). Alternate printing asterisks and underscores until N characters have been printed on that line.

On even-numbered lines, the first character will be an underscore, followed by an asterisk. Alternate printing underscores and asterisks until N characters have been printed on that line.

Sample input

```
5
```

Sample output

```
*_*_*
_*_*_
*_*_*
_*_*_
*_*_*
```

Problem 2: 2048
Run time limit: 2 seconds

Problem Description

In the popular game *2048*, you aim to merge a pair of tiles with numbers on them, and doing so adds to your score the sum of their values, and constructs a new tile from their sum. So for example, if you have two tiles with '2' on them and they merge, your score will be increased by 4, and the two '2' tiles are replaced by a '4' tile. If you have two '128's and merge them, your score will go up by 256, and there will be a 256 tile.

So from that, the question becomes what is the lowest score you can have, and still acquire the 2048 tile?

However since *2048* rose to popularity, a bunch of knock-offs have come about, one of which the maximum score you can achieve is 65536.

There are other games where you must merge more than two tiles together, say 3 tiles, and so the game goes combining 3,3,3 adds a 9 tile and gives you a score of 9, and the goal there may be (for example 177147 at 10 top tile merges).

Given a set of games, print the minimum score that can be achieved to still yield the goal tile.

NOTE: All tiles will start at the merge number. This means that for 2048 all 'spawned' digits will be '2', for 177147 the starter tiles will be '3'

Data Specification

Input

The input will start with a digit N, which denotes how many lines follow.

Then each line will have two numbers separated by a space.

The first number is the number ('Y') is the merge factor, and any number between 2 and 2,147,483,647.

The second number ('Z') is the goal tile. It may be any digit between Y and 9,223,372,036,854,775,807 (inclusive).

Output

The minimum score required to acquire the goal tile for each input question, each on their own line.

Sample input

```
3
2 2048
3 177147
78125 78125
```

Sample output

```
20480
1771470
0
```

Problem 3: Looking For Love
Run time limit: 5 seconds

Background:
Carlin is looking for love.

While browsing Reddit one day, he read a very interesting article. The article laid out an algorithm which, given the name of someone, could calculate a score indicating their compatibility. What's more, a comment from another Redditor verified the articles claims: "seems legit".

Carlin could barely contain his excitement, and got to work creating a program that could calculate the compatibility of a list of individuals, given only the name, and display the results sorted such that the name with the highest score is at the top, and the lowest score at the bottom. But before he could even fire up his favourite IDE, he was called away on secret ProgSoc business...

Can you help Carlin find love? Write a program that reads in a list of names from standard input, and outputs a ranked list of names and scores.

The Compatibility Algorithm:
The article states that a compatibility score can be determined based on the characters in the name.

The characters contribute to the score in this arrangement:
A *vowel* adds 3 to the score
A *t* adds 3 to the score
A *z* subtracts 3 from the score
A *p* adds 1 to the score
A *k* adds 2 to the score
A *s* adds 4 to the score

All other characters do not contribute to the score.

For example, the name 'carlin' breaks down to 0 + 3 + 0 + 0 +3 + 0 = 6. Because the 'a' and the 'I' are vowels hence contribute 3 each to the score, and none of the other letters contribute to the score.

Then, the number of characters in the name times 0.3 is subtracted from the current score.

Carlin has a score of 6, but has 6 letters in it. So his score of 6 becomes 4.2.

Lastly, the score is rounded down (or in maths terms, floored), and becomes the final score of 4.

In summary, the algorithm:

- Calculates a score based on the characters
- Adjusts it down based on the number of characters in the name
- And rounds it down to an integer number.

One final thing of note, the algorithm is not case sensitive, so the scoring AND the display of names should all be lowercase.

Input / Output format:
The first line your program receives from standard input is a number. This number represents the number of names N which will follow. Every N lines after that, will be a name. There will be at most 40,000 lines.

When all the lines specified by the first number have been consumed, output must be ordered by rank descending. Where the rank is the same, the names should be ordered alphabetically ascending. Names passed in are unique.

Each output line shall be in the form:
<name in lower case> :<rank>

Note that there is a SPACE between the name and the colon (:)

Sample Input:
6
Jemma
Ray
Maria
Jacinta
Bella
Christen
Sample Output:
christen :10
jacinta :9
maria :7
bella :4
jemma :4
ray :2

Notes:

- Note the order of the output.
- Names must be converted to lowercase for both display and scoring.
- The sample data will never have a duplicate name.

Problem 4: Sum of Products
Run time limit: 10 seconds

Problem Description

Jennifer has just taken a subject on binary arithmetic, and is finding that she isn't a fan of finding the *sum of products* for a particular logic expression, which is the set of all cases for which a particular logic expression will be true.

For the expression **A.B** (or "A AND B") we can create the following truth table:

A	**B**	**Output**	**Number**
0	0	0	0
0	1	0	1
1	0	0	2
1	1	1	3

For the expression **A.B**, the only 'true' case is when A = 1 and B = 1, which corresponds to 3 in the table (or 11 in binary). For this reason we say the sum of products is **F(A, B) = S{3}**

Let's look at a more complicated expression: **!A+(B.C)** (or, in words, NOT A OR (B AND C))

(NOT or **!** has the effect of inverting the value of a symbol or expression, OR or + outputs a value of 1 when either X or Y or both have a value of 1, and outputs a value of 0 when X and Y both equal 0)

A	**B**	**C**	**Output**	**Number**
0	0	0	1	0
0	0	1	1	1
0	1	0	1	2
0	1	1	1	3
1	0	0	0	4
1	0	1	0	5
1	1	0	0	6
1	1	1	1	7

True cases are 0, 1, 2, 3 and 7. Therefore the sum of products is **F(A, B, C) = S{0, 1, 2, 3, 7}**.

Write a program for Jennifer that accepts a logic expression as input and prints out its sum of products. Use a '+' character for OR and a '.' character for AND, and '!' for NOT. The characters '1' & '0' denote literals, and any upper case alphabetical character is a variable.

The order of operations is as follows: NOT > AND > OR. So the expression A+!B.C is the same as A+((!B).C).

Data Specification

Input

A line of a single two digit integer **N**, indicating how many expressions will follow.

N lines of logic expressions, consisting only of **A-J** (inclusive), **0, 1, (,), ., +, !**

Output

N lines of the sum of products of the expression, in the format **F(vars) = S{trueCases}**

Sample input

```
04
A
!B+(J.C)
B.0
(A.(B+(C.(D+(E)))))
```

Sample output

```
F(A) = S{1}
F(B, C, J) = S{0, 1, 2, 3, 7}
F(B) = S{}
F(A, B, C, D, E) = S{21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31}
```

Problem 5: Tomdex
Run time limit: 5 seconds

Problem Description

Thomas Dexter, the principal of Winter Valley High, would like to conduct a survey of all the students at his school. In particular, Mr. Dexter would like to know which names are the most common and which are the least common.

After skimming through all of the class rolls, Mr. Dexter noticed two things in common between a lot of the students' names: that there are many names that are *essentially the same but vary only in spelling*, and others that are *similar in sound*.

With this in mind, Mr. Dexter, who was a linguistics major and computer science minor at university years ago, came up with a phonetic algorithm to help classify and group names together. Proud of his achievement, Mr. Dexter named the algorithm after himself: *Tomdex*. This is a description of his algorithm.

The following letters for the purpose of the algorithm are *vowels*:

A, E, I, O, U, H, Y

All other letters are *consonants*. Consonants have numerical value, as described in step 1, vowels -- with one notable exception -- do not.

1. Take a name and replace the consonants (and initial vowel) listed below, with the following digits:

B, P -> 1 (*plosives*)
F, V -> 2 (*fricatives*)
D, T -> 3 (*dentals*)
M, N -> 4 (*nasals*)
R, W -> 5 (*rhotics*)
K, Q, X -> 6 (*velars*)
S, Z -> 7 (*sibilants*)
L -> 8 (*alveolar*)
J -> 9 (*affricate*)
first vowel in a word -> 0

The letters **C** and **G** are a bit tricky, since their pronunciations tend to vary. When encoding these letters, note the vowel that succeeds them and encode as below:

Ca, Ci, Co, Cu -> 6
Ce, Ch, Cy -> 7
C other cases -> 6

Ga, Go, Gu, Gh -> 6
Gi, Ge, Gy -> 9
G other cases -> 6

2. Remove all vowels from the word, but note their original positions in the word.

3. Looking at the remaining consonants two at a time, moving from left to right, and applying the transformations in the order listed below :-

a. If two consonants with the same number are adjacent in the word and were not separated by a vowel, only retain the first letter.

b. Two consonants with the same number originally separated by **H** are encoded as a single number.

c. Two consonants with the same number originally separated by one or more of **A, E, I, O, U, Y** are encoded twice.

Mr. Dexter hasn't touched a computer since the IBM System/360 days, so he has asked you, a lowly Computing Studies teacher at Winter Valley High, to implement his algorithm.

Data Specification

Input

Up to 1000 names, each on their own line, consisting solely of alphabetic characters (a-z and A-Z) and of a maximum length of 20 characters. The names are in no particular order. Read in names until there are no more names to read (terminated by end-of-file marker).

Output

A series of lines. On each line there will be three data elements, separated by a single SPACE character, in the following order:

1. A Tomdex code;
2. The number of names that share that code, and;
3. The names themselves sorted in ascending natural (alphabetic) order. Each name will be separated by a single SPACE.

The lines will be sorted firstly by the number of names in descending numerical order, then secondly by Tomdex code in ascending ASCII order.

Sample input

```
Mallachi
Olivia
Melissah
Cristel
Breeanah
BriAnne
Jamiroquai
Lerlinda
Bryannee
Klaneesha
Breeyanah
Schapelle
Aalivyah
Crystal
alivia
```

Sample output

```
154 4 Breeanah Breeyanah BriAnne Bryannee
082 3 Aalivyah alivia Olivia
487 2 Mallachi Melissah
65738 2 Cristel Crystal
6847 1 Klaneesha
718 1 Schapelle
85843 1 Lerlinda
9456 1 Jamiroquai
```

Problem 6: Billiard a la Descartes
Run time limit: 5 seconds

Problem description

You've played billiards before, right? If not, then surely you've played, or at the very least, watched a variant of billiards such as pool or snooker being played? You're probably familiar with the playing surface (pool table), on top of which are round, spherical objects (billiard balls), which are struck with a long stick (cue) with the intent of sinking these balls into several holes, or pockets, located at the edges of the pool table.

Now imagine a 'minimalist' pool table with no edges whatsoever. It's a planar surface that stretches out infinitely. It does have a well-defined centre, however -- let's call it the Origin. From this Origin, we can create a co-ordinate system based upon two imaginary lines, or axes, that intersect at 90-degree angles -- call them x and y.

On our boundless pool table, we have but one billiard ball and one hole in which to sink our ball. At the start of our game, both hole and ball are located at two distinct places on the table. We'll use our Cartesian-based co-ordinate system to track their position on the table.

We have a cue to strike our ball with. However, we don't have any control over how hard the cue will strike the ball, nor in which direction it will be hit. That's too easy; we'd just be aiming for the hole each time, wouldn't we?

We have been supplied with one gift from the Gaming Gods that control the table: a straight, reflective surface which we can use to redirect the ball towards the hole. Prior to the ball being struck, we will be given the initial velocity, its constant rate of deceleration and the direction, relative to the x-axis, in which the ball is struck.

Once the ball is in motion, the ball will travel in a perfectly straight line and will not deviate along its path...until it hits the reflector that we've placed in its way, where it will continue in another perfectly straight line until it comes to rest.

As you know, when a ball hits a surface, the angle at which it strikes the surface (the angle of incidence) is equal in magnitude to the angle at which it reflects off the surface (the angle of reflection). We will use this principle to guide our ball towards the hole.

Note that at all times, from the first strike to the ball coming to rest, the rate of deceleration remains constant, with or without the reflector. This means the ball will travel the same total distance.

So the two questions at hand are: where to place the reflector and how to position it, such that the ball will fall in the hole at rest. That last part of the second question is crucial: the ball will simply roll over the hole and continue on its journey, thereby missing the hole, if you position the reflector too soon along the initial path. If you position it too late, it will miss the hole entirely. Your job is to find the 'sweet spot'.

There will also be times when you won't need the reflector, either because the Gaming Gods have been kind to us and have struck the ball towards the hole with just the right amount of force (a 'hole in one') or -- far more likely -- they've been mean to us and the ball might not be struck with sufficient force to be able to reach the hole, or maybe it might roll directly over the hole and there's nothing we can do about it.

N.B. Since we will be dealing with floating point numbers, the Gaming Gods have given us one more dispensation: two numbers will be deemed equal, if the absolute value of the difference between the two numbers is less than 0.1

Data specification

Input

A series of test cases, one on each line, to be read in until there are no more cases to be read.

For each test case, there will be seven numbers separated by a space and each number shall have one decimal point. The meaning of each number, from left to right, is as follows:

1. The x co-ordinate of the starting point of the ball (-10.0 <= Xs <= -10.0)
2. The y co-ordinate of the starting point of the ball (-10.0 <= Ys <= -10.0)
3. The x co-ordinate of the hole (-10.0 <= Xt <= -10.0)
4. The y co-ordinate of the hole (-10.0 <= Xt <= -10.0)
5. The initial velocity of the ball (0.0 < u <= 100.0)
6. The rate of deceleration of the ball (-20.0 <= a < 0.0)
7. The angle at which the ball is struck relative to the x-axis (-180.0 <= θ <= 180.0)

Output

One line corresponding to each test case.

If a reflection point cannot be determined for whatever reason, print "Impossible".

If the ball rolls in the hole without the need for intervention, print "Hole in one".

If a reflection point can be found, print the following three numbers, separated by a space, each to one decimal place:

1. The x co-ordinate of the reflection point (any real value of x to one decimal place).
2. The y co-ordinate of the reflection point (any real value of y to one decimal place).
3. The angle of incidence/reflection (0.0 < α <= 90.0)

Sample input

```
0.0 0.0 3.0 4.0 10.0 -2.0 20.0
1.0 1.0 4.0 5.0 10.0 -2.0 45.0
0.0 0.0 -3.0 -4.0 2.0 -2.0 -135.0
0.0 0.0 3.0 4.0 10.0 -2.0 -90.0
0.0 0.0 25.0 0.0 10.0 -2.0 0.0
```

Sample output

```
13.5 4.9 82.5
11.6 11.5 88.0
Impossible
0.0 -10.3 84.1
Hole in one
```

Problem 7: Shopping Malls
Run time limit: 5 seconds

Problem Description

We want to create a smartphone application to help visitors of a shopping mall and you have to calculate the shortest path between pairs of locations in the mall. Given the current location of the visitor and his destination, the application will show the shortest walking path (in metres) to arrive to the destination.

The mall has N places in several floors connected by walking paths, lifts, stairs and escalators (automated stairs). Note that the shortest path in meters may involve using an escalator in the opposite direction. We only want to count the distance that the visitor has walked so each type of movement between places has a different cost in metres:

* If walking or taking the stairs the distance is the Euclidean distance between the points.

* Using the lift has a cost of 1 metre because once we enter the lift we do not walk at all. One lift can only connect 2 points. An actual lift connects the same point of different floors in the map all the points connected by a lift have the corresponding edge. So you do not need to worry about that. For instance, if there are three floors and one lift at position (1,2) of each floor, the input contains the edges (0,1,2) -> (1,1,2), (1,1,2) -> (2,1,2) and (0,1,2) -> (2,1,2). In some maps it can be possible that a lift does not connect all the floors then some of the edges will not be in the input.

* The escalator has two uses:

- Moving from A to B (proper direction) the cost is 1 metre because we only walk a few steps and then the escalator moves us.

- Moving from B to A (opposite direction) has a cost of the Euclidean distance between B and A multiplied by a factor of 3.

The shortest walking path must use only these connections. All the places are connected to each other by at least one path.

Data Specification

Input

Input contains the map of a unique shopping mall and a list of queries.

The first line contains two integers N (N <= 200) and M (N - 1 <= M <+ 1000), the number of places and connections respectively. The places are numbered from 0 to N - 1. The next N lines contain floor and the coordinates x, y of the places, one place per line. The distance between floors is 5 metres. The othertwo coordinates x and y are expressed in metres.

The next M lines contain the direct connections between places. Each connection is defined by the identifier of both places and the type of movement (one of the following: walking, stairs, lift, or escalator). Check the cost of each type in the description above. The type for places in the same floor is walking. The next line contains an integer Q (1 <= Q <= 1000) that represents the number of queries that follow. The next Q lines contain two places each a and b. We want the shortest walking path distance to go from a to b.

Output

For each query write a line with the shortest path in walked metres from the origin to the destination, with each place separated by a space.

Sample input

```
6 7
3 2 3
3 5 3
2 2 3
2 6 4
1 1 3
1 4 2
0 1 walking
0 2 lift
1 2 stairs
2 3 walking
3 4 escalator
5 3 escalator
4 5 walking
5
0 1
1 2
3 5
5 3
5 1
```

Sample output

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

Problem 8: Glass Cutting
Run time limit: 60 seconds

Problem Description

Window glass is manufactured in large rectangular sheets. An important problem in the glass business is trying to cut these sheets into the various pane sizes required by customers as efficiently as possible. PC Glass Limited is working on a new strategy to address the problem and has asked you to develop software to help. Their system is as follows:

Sizes are measured using a new unit – the ‘Glass Regular Inch’ (‘grinch’ for short – the grinch is close to an inch, but just a little smaller for easy conversion to millimetres (25 mm/grinch)).

Glass sheets are manufactured with dimensions that are always an integral number of grinches. For example, they may manufacture 100 by 200 grinch sheets.

Glass panes are sold in a set number of standard sizes (all rectangles in integral grinch sizes).

Depending on stock levels and demand, PC Glass chooses and makes regular alterations to the prices at which they sell panes of different sizes.

After setting prices, PC Glass wants software to determine an optimal cutting pattern. The software will be given a sheet size and a list of required pane sizes with prices. It must decide how best to cut one sheet into panes, so as to maximise the value of panes obtained (less the cost of the cuts). The collection of panes that results is of concern. For example, if a sheet is cut entirely into panes of one size and no panes of other sizes are produced and this happens for too long, PC Glass will simply alter the prices to favour other pane sizes and run the program again.

Glass cutting has one unusual feature that must be taken into account. The method of cutting a sheet of glass is to score (scratch) it in a straight line, and then to snap the glass along the scored line. This means that the cut must go all the way from one side of the glass to the other. Further the glass cutting tables at PC Glass only allow cuts parallel to an existing side of a piece of glass – always leading to rectangular panes.

Data Specification

Input

Input will start with a single line holding one positive integer ($0 < N < 100$), being the number of cutting problems to solve. This will be followed by data for each of the N problems. The data for a problem starts with a line of four numbers: W, H, C and S. W and H are the width and height of the glass sheet in grinches: 1 <= W, H <= 500. C is the cost of making one cut: 0 <= C <= 1000. S is the number of sizes of pane that may be cut: 1 <= S <= 20. Next are S lines, each with three numbers: w, h and p. Where w and h are the width and height of a pane and p is its price. 1 <= w <= W, 1 <= h <= H. Note that prices are positive integers - all finance is managed in integral dollar amounts.

Output

One line per problem, holding a single number – the maximum price obtainable by cutting the given sheet of glass with the given pane price structure, less the cost of making the cuts. Notes:

The cutting pattern for maximum price may not be unique.

The orientation of the glass is not important – 1 by 3 grinch pane may be cut as a 1 by 3 or as a 3 by 1.

There may be unused glass that is wasted after making cuts.

Sample input

```
4
4 3 5 1
2 1 10
3 4 5 3
3 3 10
1 2 3
1 3 4
3 4 20 3
3 3 10
1 2 3
1 3 4
3 4 1 3
3 3 10
1 2 3
```

Sample output

```
35
9
0
13
```

1. How to compile and run your programs:

Java

```
$ javac problem1.java
$ java problem1
```

N.B. the name of your class should also be `problem1`, with a single method:

```
public class problem1
{
        public static void main(String args[]) {
            // code
        }
}
```

C

```
$ gcc -lm problem1.c -o problem1
$ ./problem1
```

C++

```
$ g++ -lm problem1.cpp -o problem1
$ ./problem1
```

`-lm` links in the maths library that contains various maths-related functions. You will still need to include the appropriate header files e.g. `math.h`. Java has no command line options for maths functions, but you will need to import the appropriate class e.g. `java.lang.Math`

2. How to parse standard input in C:

How to read in an undetermined number of lines from standard input:

Basically, you set up a while loop that tests the result returned by the function used for input (`scanf(), fgets(), sscanf(),` etc.), and terminate the loop if the result returned is zero or `EOF` (end of file).

The simplest is `scanf()`, which is useful if each line of input is of known fixed format and length, e.g.

```
/* read 3 integers from each line of standard input, and print the sum for
each line */
#include <stdio.h>
int main() {
int a,b,c;
  while (scanf("%d %d %d",&a,&b,&c)!=EOF) {
        printf("%d\n",a+b+c);
  }
  return 0;
}
```

If you don't know how many items there will be on each line, use `fgets()` to read the line into a buffer, then use `strtok()` or `sscanf()` on that buffer to read in individual items, e.g.

```
/* read up to 10 integers from each line of standard input and print the sum
for each line */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main() {
char linebuf[1024];  /* A 1024 character buffer is more than enough for this
                        problem */
char* token;
int sum;

  while (fgets(linebuf, sizeof(linebuf), stdin) !=0) {
        sum=0;
        token = strtok(linebuf," \t");  /* we are using space and tab as
                                           delimiters */
        while (token != NULL) {
                sum += atoi(token);
                token = strtok(NULL, " \t");
        }
        printf("%d\n",sum);
  }
return 0;
}
```

When you are testing your program by typing from the terminal, type `control-d` to signal end of file (`EOF`).

Read the `man` pages of `sscanf(), fgets(), strtok()`, and `atoi()` (available on your machine) to understand how these system functions work, what their return values signify, and what libraries you need to include.

3. How to parse standard input in C++:

There are several ways to do this, including using the C code above. The most common way is to use `cin` and `cout`.

```
// Read in an integer, a string, and a floating point number, and print them
out
#include <iostream>
#include <string>
using namespace std;

int main() {
int x;
double y;
string s;
  cin >> x >> y >> s;
  cout << s << endl;
  cout << x << " " << y << endl;
  return 0;
}
```

You can also use `getline()`. Make sure you know how to use the delimiter argument. Depending on how you write your code, you may have to call `getline()` just to throw away the newline at the end of a line of input.

```
// reading strings from standard input with getline()
#include <iostream>
#include <string>
using namespace std;

int main () {
  string s1,s2,s3;
  getline(cin,s1,' ');
  getline(cin,s2,' ');
  getline(cin,s3);
  cout << s1 << endl;
  cout << s2 << endl;
  cout << s3 << endl;
  return 0;
}
```

How to read in an undetermined number of lines from standard input:

Basically, you set up a while loop that tests the result returned by the method `cin.operator>>`, and terminate the loop if the result returned is zero.

```
while (cin >> x) {
        // process input
}

while (getline(cin, linebuf)) {
        // process input
}
```

4. How to parse standard input in Java:

Again there are several ways to do this. Here we present only one - using the `Scanner` class.

```
import java.util.*;
import java.io.*;

public class foo
{
  public static void main (String args[])
  {
        Scanner in = new Scanner(System.in);
        String s, linebuf;

        int n = in.nextInt();
        double x = in.nextDouble();
        double y = in.nextDouble();
        char c = in.next().charAt(0);
        s = in.next(); // read in the next token (tokens are separated by
                      //whitespace by default)
        in.nextLine(); // throw away the rest of the line
        linebuf = in.nextLine();

        System.out.println(n + " " + x + " " + y + " " + c );
        System.out.print(s+" ");
        System.out.println(linebuf);
  }
}
```

No guarantee is made regarding the accuracy of information in this appendix. We hope you are already familiar with processing standard input yourself.