Manitoba High School Programming Contest

University of Manitoba

May 27, 2011 12:30-3:30 PM

Problem Packs

Notes:

- All input should be read from standard input (the keyboard) and written to standard output (the screen).
- Any programming language resources are allowed.

Problem 1 – Motion

One of the equations of motion in physics is

$$v_f^2 = v_i^2 + 2ad$$

where v_f is the final velocity of an object (m/s), v_i is its initial velocity (m/s), a is its acceleration (m/s^2) and d is the distance it travels (m). So this formula allows you to calculate how fast an object is going at an ending point if you know what speed it began at, the distance from the starting to the ending point and the (constant) acceleration from the starting to the ending point.

Use this formula to compute the final velocity of an object given the other three values.

Input

Your program should accept three quantities on separate lines of input: first the initial velocity (m/s), then the acceleration (m/s^2), and then the distance (m).

Output

Your program should then output the final velocity in the following format:

`"The final velocity is X m/s."`

where `X` is a decimal number. The number may have any number of decimal places.

Sample Input	Sample Output
1 1 1	`The final velocity is 1.7320508075688772 m/s.`

Judging Data for Problem 1

Note: since this problem does not involve any looping, your program will be run separately on each of the five input cases listed below.

Case 1:

```
1
1
1
```

Case 2:

```
0
0
2
```

Case 3:

```
0
9.806
500
```

Case 4:

```
-5
9.806
100
```

Case 5:

```
30
1.62
1000
```

Problem 2 – Air travel

How far apart are two airports? To find out, we use the great-circle formula, given below. In this question, the positions of the airports will be given by latitude and longitude. Latitude is the angle between -90 and 90 degrees that gives the angle between that point and the equator, while longitude is the angle between 180 and -180 degrees that gives the angle between the point and a fixed principal meridian.

Consider two points on a sphere with radius r: point one is at latitude A_1 and longitude B_1, while point two is at latitude A_2 and longitude B_2. The distance over the surface of the sphere between the points is given by

$$d = r * \arccos (\sin (A_2) * \sin (A_1) + \cos (A_2) * \cos(A_1) * \cos (B_2-B_1)).$$

In this question, we are concerned with distances between points on Earth, which is (almost) a sphere with radius is 6371 km.

Input

Each line of input will list two airports by their IATA airport code, followed by their latitude and longitude. The latitude and longitude will be given as whole degrees and may be positive or negative.

Output

For each pair of airports, give the distance between them in kilometres, formatted as follows

```
Distance from XXX to YYY is ZZZZ km.
```

In this printout, XXX and YYY are the IATA airport codes. The distance should be the closest integer to the distance given by the formula.

Sample Input	Sample Output
YDN 51 -100 YWG 49 -97	Distance from YDN to YWG is 309 km.
YDN 51 -100 XTL 58 -98	Distance from YDN to XTL is 789 km.
YDN 51 -100 GOH 64 -51	Distance from YDN to GOH is 3160 km.
HRB 45 126 GRU -23 -46	Distance from HRB to GRU is 17463 km.

Judging Data for Problem 2

Note: for this and all further problems, the inputs are all given as a single input. Your program should accept all of the input during one run without restarting.

```
YDN 51 -100 YWG 49 -97
YDN 51 -100 XTL 58 -98
YDN 51 -100 GOH 64 -51
HRB 45 126 GRU -23 -46
FOO -1 135 BAR 0 -176
BAZ -1 -162 BAT -20 -48
JAV 61 -51 ASE 39 -105
VIS 36 -119 UAL -10 22
BAS -6 155 ICY 59 -141
```

Problem 3 – Stock Tracker

A friend of yours has found a group of stocks that she really likes. They have very unusual behaviours: each day, each of the stocks either moves up 10 points, stays the same, or moves down 10 points. They never do anything else.

Your friend is somewhat risk averse. After she buys a stock, she sets a value *p* that tells her when she will sell. If the price drops *10*p* points from her initial purchase price, she will sell. That is, if the price is ever equal to *10*p* points less than the initial purchase price, your friend will sell then.

Write a program that simulates your friend's behaviour. You are given a sequence of symbols `U` (stock moves up), `S` (stock stays the same) and `D` (stock moves down) representing the action of a stock over a period of days after purchasing, as well as a value of *p*, and then would like to report either

1. how many days after purchasing the stock did your friend sell the stock (due to dropping *10*p* points from the initial purchase price).
2. or, if she did not sell, what the change in value of the stock was at the end of the period.

Input

Each line consists of a number `p` and a sequence of symbols `D`, `U`, `S` representing down, up and stay.

Output

For each line, first output "`Stock N: `" where `N` is the number of the stock in the file, starting at `N=1`. On the same line, output either "`Sold after X day(s).`", "`Price is Y points higher than at purchase.`", "`Price is Y points lower than at purchase.`" or "`Price is unchanged.`" where `X` and `Y` are positive numbers.

Sample Input	Sample Output
3 UUDUSSUSUU 2 DDUDSSSSS 3 DDUDSSSSS	Stock 1: Price is 50 points higher than at purchase. Stock 2: Sold after 2 day(s). Stock 3: Price is 20 points lower than at purchase.

Judging Data for Problem 3

3 UUDUSSUSUU
2 DDUDSSSSS
3 DDUDSSSSS
3 DDUDDSSUD
3 UUDDDDSSS
4 UDDUDDUDDUDD
4 UDDUDDUDDUUD
2 SSSSSSSSSSSS
2 UDUDUDUDUDUDUD
2 UUDDUUDDUUDDUUDD
2 UUUUUUUUUUUUUUUU
1 UUUUUUUUUUUUUUUU
1 UUUUUUDDDDDDUUUUUUDDDDDDUUUUUUDDDDDDU
5 UDSUSUSUDDUSUSDUSDUSDDUSUSUSSSUUDSUSUDUD
5 UDSSUSUSUSSDUSUSDUSUSSDDUSSUDUUDSUSDUUDUD
5 DUSDSUUUSDDDUUSUDDUUDDDDDUSUSDUSDUUSSDUUUS
4 DUSDSUUUSDDDUUSUDDUUDDDDDUSUSDUSDUUSSDUUUS
3 DUSDSUUUSDDDUUSUDDUUDDDDDUSUSDUSDUUSSDUUUS
2 DUSDSUUUSDDDUUSUDDUUDDDDDUSUSDUSDUUSSDUUUS
1 DUSDSUUUSDDDUUSUDDUUDDDDDUSUSDUSDUUSSDUUUS

Problem 4 – Slitherlink

Slitherlink is a two-dimensional puzzle in which a partially filled grid of numbers is given. The goal is to find a path through the grid that does not intersect itself, starts and ends at the same point, and where the number of edges visited by the path of any square in the grid is equal to the number written in the center of that square, if there is one. (If there is no number, the path can pass through any number of edges.)

Here is an example: the puzzle is given on the left, and a solution is shown on the right by the dark line.

		3	3	3	2
3				1	
		3	2		
1		2		2	
3				3	2
		2	1		

		3	3	3	2
3				1	
		3	2		
1		2		2	
3				3	2
		2	1		

Solving slitherlink with a computer program is hard. But in this question, we are asking for you to help *design* the problems. You are given the path, and are asked to return the grid of all possible numbers that we could give as part of the puzzle. Clearly, we wouldn't give them all as part of the puzzle, but we want to know what they all are.

Input

For each puzzle, you are given first a dimension n on a single line, then, then a 2n+1-by-2n+1 grid made up of the following characters: spaces, vertical bars (|), underlines (_) and periods (.).

The grid has a closed non-intersecting path represented by the bars, underlines and periods. The bars and underlines represent a vertical or horizontal move, while the period represents a corner. Spaces mean that no path passed through the point. There are always enough spaces to fill the 2n+1-by-2n+1 grid. (The path is written as a 2n+1-by-2n+1 grid because corners take up a single character. However, the path really passes through the lines of an n-by-n grid.)

There are no blank lines between grids. The final line of the input file is a zero. This input should not be processed.

Output

Your program should output the same path with all numbers filled in. Notice that this means you will write n^2 numbers.

Sample Input	Sample Output

Sample Input:

```
3
._._._.
|     |
. ._. .
| | | |
._. ._.
7
._._.   ._.
|   |   | |
._. .   . ._.
  | |   |   |
._. ._._.   ._.
|             |
._. ._._.   ._.
  | |   |   |
._. .   .   .
|   |   |   |
.   .   .   ._.
|   |   |     |
.   .   ._._._.
|   |
._._.
0
```

Sample Output:

```
._._._.
|2 2 2|
. ._. .
|3|3|3|
._. ._.
 1 0 1
._._.   ._.
|3 2|1 1|3|2 0
._. .   . ._.
 3|2|2 2|1 2|2
._. ._._.   ._.
|3 0 2 2 0 0 3|
._. ._._.   ._.
 3|2|2 2|1 1|2
._. .   .   .
|2 1|1 1|1 1|2
.   .   .   ._.
|1 1|1 1|2 1 3|
.   .   ._._._.
|2 2|1 0 1 1 1
._._.
```

Judging Data for Problem 4

```
1
._.
| |
._.
2
._.
| |
._.


2
._._.
|   |
.   .
|   |
._._.
3
  ._.
  | |
  ._.




3
    ._.
    | |
    ._.




3
._._.
|   |
._. .
  | |
  . ._.
  |   |
  ._._.
```

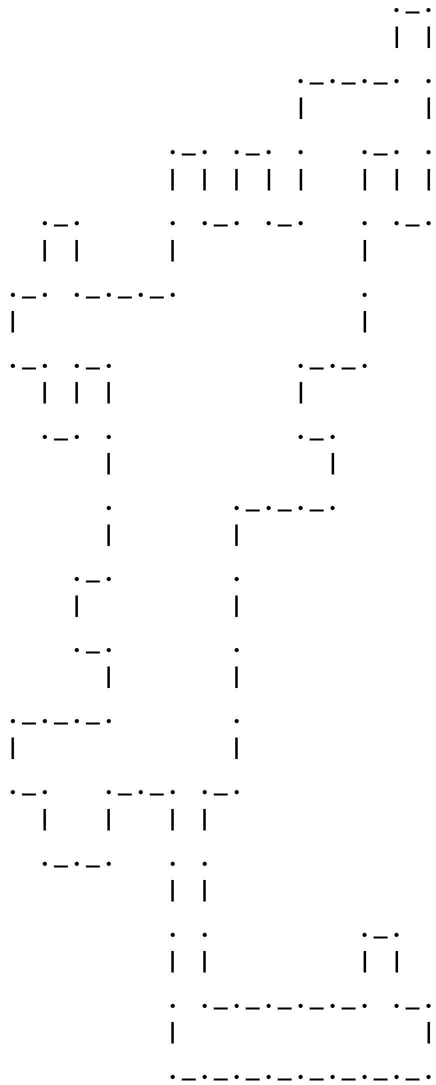
(continued on next page)

3
```
._._._.
|     |
. ._. .
| | | |
._. ._.
```

4
```
._. ._.
| | | |
. ._. ._.
|       |
._.     .
  |     |
  . ._. .
  | | | |
  ._. ._.
```
7
```
._._.   ._.
|   |   | |
._. .   . ._.
  | |   |   |
._. ._._.   ._.
|             |
._. ._._.   ._.
  | |   |   |
._. .   .   .
|   |   |   |
.   .   .   ._.
|   |   |     |
.   .   ._._._.
|   |
._._.
```

(continued on next page)

15



0

Problem 5 – Ten Pin Bowling

Scoring (ten pin) bowling is a complex affair. In each frame, ten pins are set up and the bowler has two throws to knock down as many pins as possible. The outcomes are:

- open frame: pins are left standing at the end of the frame.
- spare: all ten pins are knocked down using two throws.
- strike: all ten pins are knocked down using one throw.

Each game consists of ten frames.

The bowler scores one point for each pin knocked down, but is awarded bonus points in the case of a strike or spare. In a strike, the bowler is given ten points plus the number of pins knocked down in the next two throws. For a spare, the number of pins knocked down in the next throw is given as bonus points, in addition to the ten points earned for the spare. Strikes are denoted by X and spares by /. So if the sequence of throws in the first three frames was

X 3 / 9 0

then the score at the end of the three frames would be

- frame 1 (strike): 10 points (all ten pins knocked over) + 3 + 7 (bonus for next two throws)
- frame 2 (spare): 10 points (all ten pins knocked over) + 9 (bonus for next throw)
- frame 3 (open): 9 points
- total: 48 points

On the last (tenth) frame, a strike or spare may yield extra throws: if the bowler gets a strike on the last frame, they throw two extra balls for bonus points. If the bowler gets a spare on the tenth frame, one extra ball is thrown. No bonus points are given for the extra balls, so if one of the extra balls is a strike, no additional throws are given. In this way, a game of bowling will always (thankfully) end. Note that the extra balls count towards the bonus only, and are not themselves added to the score. So if a bowler has two extra throws, and scores 8 points on those two throws, their final score is increased by 8 (not 16).

Write a program that accepts a sequence of frame scores and gives the total score.

Input

Each line of input gives the sequence of frame scores for one game. The game is a valid list of ten frames plus extra balls if any. The frames are not separated by any indicators, but should be read as frames in the unique way. Characters are separated by a single space.

Output

Output the score for each game on its own line.

Sample Input	Sample Output
9 / X 1 2 X X 9 / 1 8 3 3 3 3 3 3	123
9 / X 1 2 X X 9 / 1 8 3 3 3 3 3 / X	137
9 / X 1 2 X X 9 / 1 8 3 3 3 3 X 7 2	136

Judging Data for Problem 5

```
9 / X 1 2 X X 9 / 1 8 3 3 3 3 3 3
9 / X 1 2 X X 9 / 1 8 3 3 3 3 3 / X
9 / X 1 2 X X 9 / 1 8 3 3 3 3 X 7 2
X X X X X X X X X X X X
X X X X X X X X X X 0 0
X X X X X X X X X 9 3
X X X X X X X X X 3 / X
X X X X X X X X X 4 / 5
X X X X X X X X 9 / X 7 2
X X 3 / X X X 5 / X 9 / X 7 2
X X 3 / X X X 5 / X 9 / X 7 /
1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9 / 0 / X
1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9 / 0 0
1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9 / 0 1
0 9 1 8 2 7 3 6 4 5 5 4 6 3 7 2 8 1 9 0
0 7 2 5 3 4 3 3 3 2 1 7 6 2 2 4 4 4 4 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

Problem 6 – Chasing JSON

JSON stands for JavaScript Object Notation. It's a way to store structured data that in a file. Here's an example of JSON-formatted data that lists a student, their student number, program and address:

```
{ firstName : Stu , lastName : Dent , number : 123456 , program : {
major : COMP , minor : ENGL } , address : { number : 9001 , streetName
: Leet , streetType : St , city : Winnipeg } }
```

A collection of data formatted in this way will be called a *JSON object*. For the purposes of this problem, there will be three types of structures in JSON objects:

1. A value, which is either a string or another object (see point 3). A string does not contain spaces and is not enclosed in quotation marks. It is case-sensitive.
2. A name-value pair which is a name followed by a colon and then a value. The name is a unique identifier that does not contain spaces.
3. An object is a list of name-value pairs (which are separated by commas) enclosed with braces.

Notice that this means that a single JSON object may have other objects nested inside them. For instance, the 'student' object in our example has objects for both their program of study and their address nested inside it.

Write a program that is able to determine whether two JSON objects represent the same information. The two objects may have information in different orders, but their key-value pairs must all be identical in both objects. For instance, the object

```
{ lastName : Dent , program : { minor : ENGL , major : COMP } , address
: { number : 9001 , streetName : Leet , streetType : St , city :
Winnipeg } , number : 123456 , firstName : Stu }
```

is equal to the object in the first example: it has all the same key-value pairs as the first example.

Input

Each JSON object is given on a separate line of input, and should be processed as pairs beginning from the first and second objects. A blank line will appear between pairs. All tokens (non-white space strings) on a line are separated by a single space from each other. All input is case-sensitive. Input is in the format described above.

Output

For each pair, write "`Pair X`" on one line, where X is a number starting at 1 for the first pair. On the next line, if the two objects in a pair are equal, write "`The two objects are equal.`", otherwise write "`The two objects are not equal.`".

<table>
<tr><th>Sample Input</th><th>Sample Output</th></tr>
<tr><td><pre>{ firstName : Stu , lastName : Dent , number : 123456 , program : {
major : COMP , minor : ENGL } , address : { number : 9001 ,
streetName : Leet , streetType : St , city : Winnipeg } }
{ lastName : Dent , program : { minor : ENLG , major : COMP } ,
address : { number : 9001 , streetName : Leet , streetType : St ,
city : Winnipeg } , number : 123456 , firstName : Stu }

{ firstName : Stu , lastName : Dent , number : 111111 , program : {
major : COMP , minor : ENGL } , address : { number : 9001 ,
streetName : Leet , streetType : St , city : Winnipeg } }
{ lastName : Dent , program : { minor : ENLG , major : COMP } ,
address : { number : 9001 , streetName : Leet , streetType : St ,
city : Winnipeg } , number : 123456 , firstName : Stu }

{ firstName : Stu, lastName : Dent , major : COMP }
{ name : { first : Stu , last : Dent } , major : COMP }</pre></td><td><pre>Pair 1
The two objects are
equal.
Pair 2
The two objects are not
equal.
Pair 3
The two objects are not
equal.</pre></td></tr>
</table>

Judging Data for Problem 6

```
{ x : y }
{ x : z }

{ x : y }
{ x : y }

{ x : y }
{ z : y }

{ x : a }
{ foo : aa }

{ foo : aa }
{ x : a , y : b }

{ x : a , y : b }
{ x : a , z : b }

{ foo : a }
{ foo : bb }

{ foo : aa , bar : bb }
{ foo : aa , bar : bb }

{ foo : aa , bar : bb }
{ baz : aa , bar : bb }

{ foo : aa , bar : bb }
{ bar : aa , foo : bb }

{ foo : aa , bar : bb }
{ bar : bb , foo : aa }

{ foo : aa , bar : { x : y , z : w } }
{ foo : aa , bar : { x : y , z : w } }

{ foo : aa , bar : { z : w , x : y } }
{ foo : aa , bar : { x : y , z : w } }

{ a : { b : { c : { d : { e : { f : g } } } } } }
{ a : { b : { c : { d : { e : { f : g } } } } } }

{ a : { b : { c : { d : { e : { f : g } } } } } }
{ a : { b : { c : { d : { e : { f : h } } } } } }

{ a : { b : { c : { d : { e : { f : { g : { h : { i : { j : { k : { l : { m : { n : o , p
: q } } } } } } } } } } } } } }
{ a : { b : { c : { d : { e : { f : { g : { h : { i : { j : { k : { l : { m : { n : o , p
: q } } } } } } } } } } } } } }

{ a : { b : { c : { d : { e : { f : { g : { h : { i : { j : { k : { l : { m : { n : o , p
: q } } } } } } } } } } } } } }
{ a : { b : { c : { d : { e : { f : { g : { h : { i : { j : { k : { l : { m : { p : q , n
: o } } } } } } } } } } } } } }
```

(continued on next page)

```
{ a : { b : { c : d } , e : { f : g } } , h : { i : { j : k } , l : { m : n } } }
{ h : { l : { m : n } , i : { j : k } } , a : { e : { f : g } , b : { c : d } } }

{ a : { b : { c : z } , e : { f : g } } , h : { i : { j : k } , l : { m : n } } }
{ h : { l : { m : n } , i : { j : k } } , a : { e : { f : g } , b : { c : d } } }

{ a : { b : { c : d } , f : g  } , h : { i : { j : k } , l : { m : n } } }
{ h : { l : { m : n } , i : { j : k } } , a : { e : { f : g } ,  c : d  } }
```
(no blank line at end of file)