Twenty-Ninth Annual

Willamette University—TAO Foundation High School Programming Contest

Saturday 14 March 2015

OFFICIAL CONTEST TEST DATA

COURTESY COPY FOR TEAMS—OK TO DISTRIBUTE!!

General advice to judges

- always read the problem before testing a program (if you haven't read the problem previously)
- allow for slight variations in how output is presented, but not for substantive issues in the answer (if you're not sure which is which, ask the Contest Director, i.e., Fritz)
- **allow for reasonable user interfaces,** as long as they are clearly labeled, or prompt appropriately for inputs
- remember to include any promised sentinel values (e.g., 0, -1, XX, a blank line) to terminate input
- if a solution fails, make some effort to see why (hit extra carriage returns, etc.—read the code!)
- in general, we don't give hints about why a solution failed, but in persistent "hard luck" cases, we will
- if a program runs correctly for all test inputs, but you suspect it is still wrong, we can add new test data
- remember, the contest is supposed to be fun (and educational)!

1. Penelope's Phone Peril *(uppercase letters, sep. by spaces, **term. blank line**, series of 10-digit nums, one per line)*

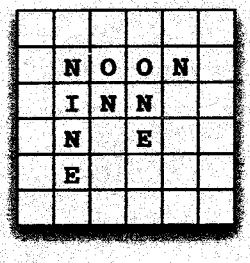
```
Input:   HELP PIZZA SOLO HAN SHOT FIRST    (= 4357 – 74992 – 7656 – 426 – 7468 – 34778)
         1237499200
Output:  PIZZA
         1230014357
Output:  HELP
         4261110000
Output:  HAN
         1234554321
Output:  No codewords found                (or similar message)


Input:   A PINEAPPLES                      (= 2 — 7463277537)
         5033706165
Output:  No codewords found                (or similar)
         2000000000
Output:  A
         7463277537
Output:  PINEAPPLES
```

```
Input:   CART ART ARTSY                    (= 2278 — 278 — 27879)
         0227879879
Output:  CART ART ARTSY                    (any order of output is OK!)
         2272272272
Output:  No codewords found                (or similar)
```

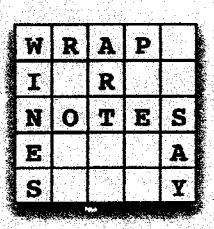
2. Letter *scramble!* *(board size = number, all uppercase words sep. by single space, all on one line; words MUST run Left-Right or Top-Down—no diagonal, bottom-up or right-left!)*

```
Input:   8
         CATTLE CARS TABLE
Output:  CATTLE at (4,7) vertical          (many possible placements—check output!)
          CARS   at (6,8) vertical
          TABLE  at (4,5) horizontal
```

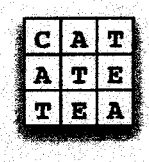
```
Input:   6
         NOON INN NINE ON ONE
Output:  NOON at (2,5) horizontal
          INN   at (2,4) horizontal
          NINE at (2,5) vertical
          ON    at (3,5) vertical
          ONE   at (4,5) vertical
```




```
Input:   5
         SAY NOTES ART WRAP WINES
Output:  SAY    at (5,3) vertical
         NOTES  at (1,3) horizontal
          ART    at (3,5) vertical
          WRAP   at (1,5) horizontal
          WINES  at (1,5) vertical
```




```
Input:   3
         CAT ATE TEA
Output:  CAT at (1,3) horizontal           (or vertical)
          ATE  at (1,2) horizontal         (or (2,3) vertical)
          TEA at (1,1) horizontal          (or (3,3) vertical)
```




```
Input:   5
         I TO ARCHAEOPTERYX                (doesn't fit—mentioned in contest problem!!)
Output:  No solution is possible
Input:   2
         IT OK                             (just no way to do it—so, IT NOT OK! :) )
Output:  No solution is possible
```

3. Fraction compaction *(≤ 10 fractions per line, all positive INPUT, spaces* except *in fractions, LOWEST TERMS)*

```
Input:    2/3  +  4/5  *  1/2  -  1/4
Output:   29/60                                   (see contest problems)

Input:    12/3  *  7/8  -  133/11  *  4/6
Output:   -63/11

Input:    4/9  -  11/13  *  47/12  +  8/3 * 1/2
Output:   1535/2808

Input:    28/4  /  7/7  /  41/12  -  3/7  /  9/5
Output:   775/861
```

4. Goldilocks and the binary digits *(repeated integers, < one billion, descriptive output in English)*

```
Input: 292
Output: Too light          ( = 100100100 )

Input: 0
Output: Too light          ( = 0 )

Input: 1
Output: Too heavy          ( = 1 )

Input: 2
Output: Just right         ( = 10 )

Input: 133
Output: Too light          ( = 10000101 )

Input: 170
Output: Just right         ( = 10101010 )

Input: 195
Output: Just right         ( = 11000011 )
```

5. Pie-cutting conundrum *(two positive integers; English message output with # of slices, size of cutter)*

```
Input:    40 7
Output:   Cut 7 pies with the 3-way cutter, into 6 slices each.

Input:    27 2
Output:   Cut 2 pies with the 5-way cutter, into 15 slices each.

Input:    3 1
Output:   Cut 1 pies with the 3-way cutter, into 3 slices each.
```

5. Pie-cutting conundrum *(continued)*

```
Input:    100 8                                          (oops: ambiguity here!)
Output:   Cut 8 pies with the 3-way cutter, into 12 slices each.
    OR:   Cut 8 pies with the 4-way cutter, into 12 slices each.
```

6. Pie-packing puzzler *(two positive integers n & m, then m lines of floating-point pairs, APPROX. OK)*

```
Input:    12      3
          0.4     2.0
          0.2     3.5
          1.1     1.5
Output:   Total weight = 2968.812 pounds          (OK if shorter, or just number)

Input:    2       2
          1.0     1.0
          1.0     1.0
Output:   Total weight = 314.159 pounds           (sanity check, 100π)

Input:    10      4
          0.3     1.5
          0.2     3.0
          1.0     1.5
          2.0     0.8
Output:   Total weight = 3259.40 pounds           (OK if just approximate)
```

7. Piling on the dominoes *(grid size integer, then sequence of "dotted pairs", multi-line;*

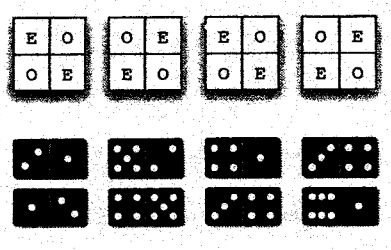
output is several grids, bottom up — MANY SOLUTIONS POSSIBLE!)

```
Input:    3                                                (from the problem set)
          2.1  1.3  4.5  6.4  3.3  0.2
Output:   3 1 6       (bottom layer)
          2 _ 4
          1 4 5

          _ _ 3       (top layer)
          _ _ 3
          2 0 _
```

(2 x 2 grid: highly constrained—but can pile high, too!!)

Input:
```
2
3.4  1.4  4.5  1.2  6.1  2.5  4.3  2.1
```

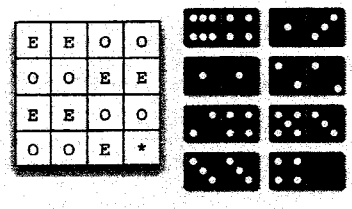
Output:
```
2 1          (bottom layer)
1 2

5 2          (next layer)
4 5

4 1          (next layer)
3 4

3 4          (top layer)
6 1
```



(possible as one layer ... but many variations in one or more layers)

Input:
```
4
6.4  0.4  3.3  1.3  5.3  4.2  1.1  2.2
```

Output:
```
6 4 1 3
1 1 2 2
2 4 5 3
3 3 4 0
```



8. **Nested storage** *(testing these may be easiest using some cut up index cards or similar physical media)*

Input: `1 3  2 4  4 2  2 2  2 2  5 4` *(from contest problem)*

Output: *(many ways to do this!!!)*
```
5 by 4  at  (0,0)
4 by 2  at  (0,0)
4 by 2  at  (0,2)
2 by 2  at  (1,2)
2 by 2  at  (1,2)
1 by 3  at  (4,0)
```

Input: `3 2  2 3  4 3  3 3  6 7  2 4`

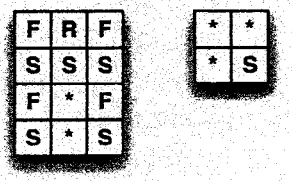
Output: *(many ways to do this!!!)*
```
6 by 7  at  (0,0)
3 by 3  at  (0,0)
3 by 2  at  (0,0)
2 by 3  at  (3,0)
2 by 4  at  (4,3)
4 by 3  at  (0,3)
```

Input: `6 6  3 3  1 4  5 7` *(no good box to put on the bottom)*

Output: `No solution possible.` *(or similar message)*

9. Terrain-spotting *(2 integer dimensions, plus data; repeat)*

```
Photo size?   7 3
Features?     RF*RRSFFRFSFFRFFRF*SF

Pattern size?   3 2
Features?       F*RFS*

Matches at:
     position 3 1
     position 5 2


Photo size?   3 4
Features?     FRFSSSF*FS*S

Pattern size?   2 2
Features?       ***S

Matches at:
     position 1 1
     position 2 1
     position 1 3
     position 2 3


Photo size? 3 2
Features?   F*RFS*

Pattern size? 1 1
Features?   *

Matches at:
     position 1 1    position 2 1    position 3 1
     position 2 1    position 2 2    position 3 2
```

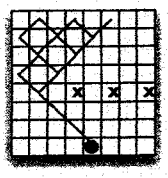


10. Checker challenge *(input is a series of integer pairs; first is start; terminate with 0 0 if requested)*

```
Input:     5 4
           4 3
           2 7
           6 7                          — start position, plus 3 blocks

Output:    LLRL
           LLRR
           LRLL
           LRLR
           RLLL
           RLLR
           RRRL


Input:     1 1;  2 2                    — totally blocked off

Output:    No paths!
```

```
Input:     5 1;  4 4;  6 4;  8 4
```
— right side mostly blocked; all paths start **LLL**

```
Output:    LLL-LRLR
           LLL-LRRL
           LLL-LRRR

           LLL-RLLR
           LLL-RLRR
           LLL-RRLL
           LLL-RRLR
           LLL-RRRL
           LLL-RRRR
```

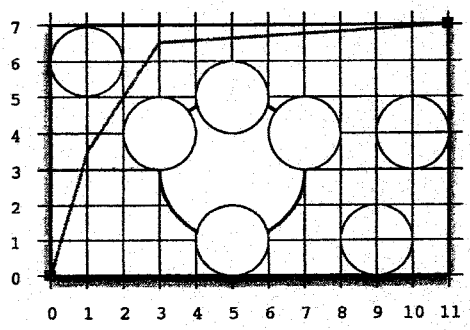


```
Input:     4 7
```
— no blocks, but short paths

```
Output:    L
           R
```

11. Zombie zig-zag *(two floats on first line, three thereafter; smidgen extra radius for edges of field)*

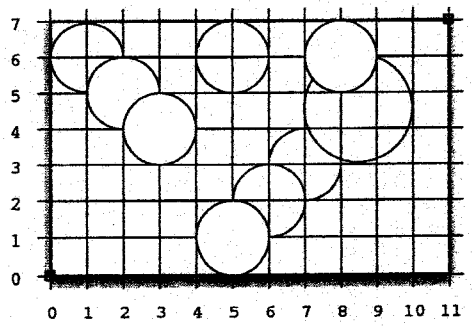
```
Input:  11.0   7.0
         1.0   6.0   1.01
         3.0   4.0   1.01
         5.0   3.0   2.01
         5.0   5.0   1.01
         5.0   3.0   1.01
         5.0   1.0   1.01
         7.0   4.0   1.01
         9.0   1.0   1.01
        10.0   4.0   1.01
```




```
Output:  0.0   0.0
         1.0   3.5
         3.0   6.5
        11.0   7.0
```

(this is just an example—any path that starts at (0,0) and ends at (11,7) and avoids circles is OK; but must also stay in rectangle)

```
Input:  11.0   7.0
         1.0   6.0   1.01
         2.0   5.0   1.01
         3.0   4.0   1.01
         5.0   6.0   1.01
         5.0   1.0   1.01
         6.0   2.0   1.01
         7.0   3.0   1.01
         8.0   6.0   1.01
         8.5   4.5   1.51
```



(careful—that's a 1.51)

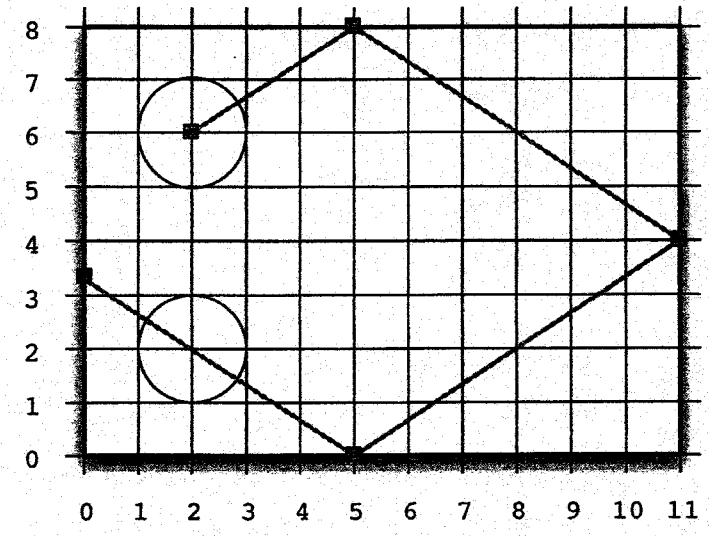
```
Output:  Oh no! Eaten by zombies!
```
(or similar "message of despair' :))

12. Operator fixer-upper *(spaces between ops, nums & vars!! but not parens; output must be fully paren'ed!!)*

```
Input:  (+ (* x 3) (- (* y 5) (/ x 7)))          (from problem set)
Output: ((x * 3) + ((y * 5) - (x / 7)))

Input:  (* (+ (* x y) (/ x 3)) (- 5 y))
Output: (((x * y) + (x / 3)) * (5 - y))

Input:  (+ (- (* (/ (+ x 1) 2) 3) y) z)          (left-nested)
Output: (((((x + 1) / 2) * 3) - y) + z)

Input:  (+ 1 (* 2 (- x (/ y 3))))                (right-nested)
Output: (1 + (2 * (x - (y / 3))))

Input:  (+ x y)                                  (short!)
Output: (x + y)
```

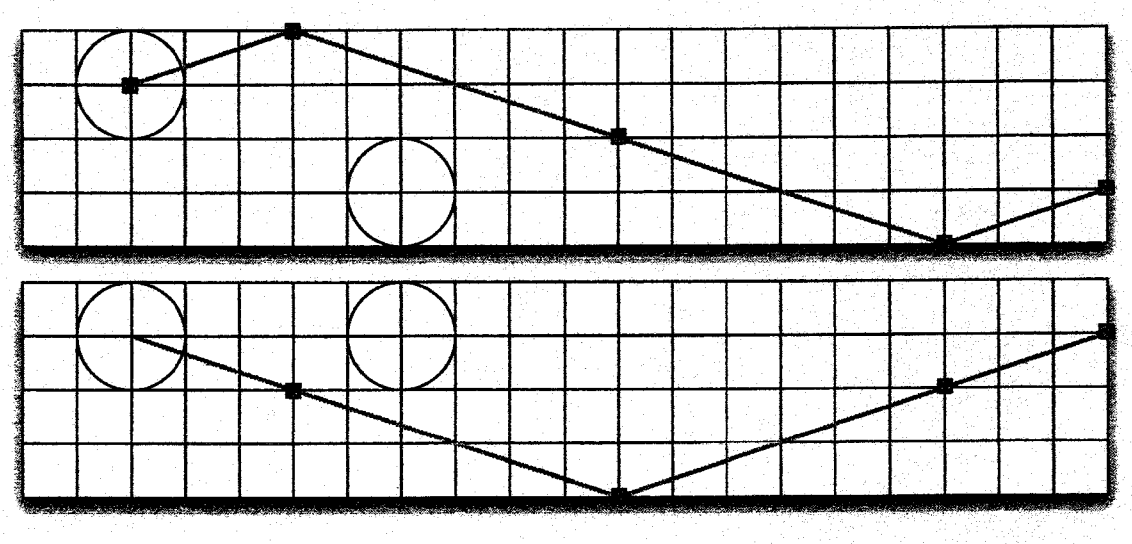
13. Spaceball! *(lots of floats: 3 on 1st line, then 4, then 4 again, then 3—see problem description!)*

```
Input:   11.0   8.0   6.0                 (game room)
          2.0   6.0   3.0   1.0           (thrower)
          2.0   2.0   3.0   1.0           (opponent)
          5.0   8.0   3.0                 (1st bounce—flat / 2D!)

Output:   5.0   8.0   3.0                 (echo 1st bounce)
         11.0   4.0   3.0                 (2nd bounce)
          5.0   0.0   3.0                 (3rd bounce)

Ball strikes opponent!
```



13. Spaceball! *(continued)*

```
Input:  20.0   4.0   4.0                 (game room)
         2.0   3.0   3.0   1.0           (thrower)
         7.0   1.0   3.0   1.0           (opponent)
         5.0   4.0   2.0                 (1st bounce)

Output:  5.0   4.0   2.0                 (echo 1st bounce)
        11.0   2.0   0.0                 (2nd bounce)
        17.0   0.0   2.0                 (3rd bounce)
        20.0   1.0   3.0                 (4th bounce)

        Ball goes out of play!
```



14. Text expander *(positive integer width; "dictionary"; blank line; text-on-a-line; be careful re blank spaces!!* ***blanks may be trimmed in output to wrap & justify!!)***

Input:

```
20
lol   laughing out loud
brb   be right back
:)    *smile*
<3    *heart*

Hey gramps! Great joke -- lol! Mom's calling, brb!
```

Output:

```
Hey gramps! Great
joke -- laughing out
loud! Mom's calling,
be right back!

12345678901234567890
```

(they don't have *to print this—just for us!)*

14. Text expander *(continued)*

Input:

```
15
aa    aardvark
xy    Cartesian
pi    3.1415926
:{)   *moustache*
oo    *infinity*

aa with a :{) is xy by pi to the oo
```

Output:

```
aardvark with
a *moustache*
is Cartesian
by 3.1415926
to the
*infinity*

1234567890123
```

(again, they don't have *to print this)*

Input:

```
6
a   a
b   b
c   c
d   d
e   2.718
f   f
g   groovy

a b c d e f g  a b c
```

Output:

```
a b c
d
2.718
f
groovy
a b c

123456
```