Irish Collegiate Programming Contest 2011

Problem Set

University College Cork ACM Student Chapter

March 26, 2011

Contents

Instructions

Rules

- Once the contest begins teams may not confer with any other person, including their coach.
- All mobile phones, laptops and other electronic devices must be powered off and stored away for the duration of the contest.
- Teams may confer privately in one of the supervised spaces outside the labs. Please let an organiser know if you need to do this.
- The only networked resource that teams are permitted to access is the submission system.
- If a team discovers an ambiguity or error in a problem statement they should tell an organiser. If the organisers agree that an ambiguity or error exists, a clarification will be issued to all teams.

Testing and Scoring

- When teams submit a solution the submission time will be recorded and the solution will be queued for testing.
- A problem will be tested against 10 test cases, the first two of which are given to you on problem sheet.
- A solution will be accepted if it compiles on the test environment.
- If a solution is submitted while an earlier solution to the same problem is still in the queue, the earlier submission will not be tested. The earlier submission will be marked with the message: ‘Old, not going to be tested’.
- All accepted problems will be scored out of 10 points, one for each test case that returns the correct output.
- You will not be told the points scored until the end of the contest.
- In the event of a tie on points, the winning team will be the one with the shortest amount of time between the contest start and their last solution that increased their score.

Submission Instructions

- The submission system URL is: http://4c128.ucc.ie/
- Your password will be provided by the organisers. Please notify an organiser if you unable to log in.
- Submissions should consist of a single source file.
- To submit, click the ‘Submit a Solution’ link, complete the submission form, upload your source file, and click save.
- Your submission should now be listed in your submission queue.
- Submission input is read from standard input and submission output should be written to standard output.
- To teams submitting Java solutions, be aware the testing script will be renaming your file to Pn.java (where n is the problem number) and as such the main class for problem 1 would be defined as:

  ```
  public class P1 {
  ...
  }
  ```

  and will be called by :

  ```
  java P1 < input-file
  ```

1 Emirps

Task An emirp is a prime number that is also prime when its digits are reversed, and that is also not a palindrome. For instance, 13 is an emirp because its reversal, 31, is also prime; 23 is not an emirp, even though it is prime, because its reversal, 32, is not prime; and 101 is not an emirp, even though it is prime, because it is a palindrome. Given two numbers N and M, your task is to find the list of emirps that occur between N and M inclusive.

Input Two numbers N and M (0 < N < M < 1,000,000) on the same line separated by a space, followed by a newline.

Output A space separated list of emirps between N and M inclusive, followed by a newline.

Sample Input Two separate sample cases:

```
1 20
```

```
50 80
```

Sample Output The corresponding sample output:

```
13 17
```

```
71 73 79
```

2 Base Addition

Task People are more familiar with addition in the decimal (base 10) numeral system, but addition is possible in number systems of any base, including binary (base 2), octal (base 8) and hexadecimal (base 16). Given 3 numbers, s1, s2, s3, where the numeral system is not specified, the expression s1 $+$ s2 $=$ s3 may be correct in some numeral systems, but not in others.

Input Input is 3 strings s1, s2, s3 separated by spaces. These strings represent numbers in an unknown numeral system (unknown base). Each string will be at most 20 characters, and will consist of digits & upper-case letters with ASCII value ordering (0, 1, 2, 3, . . . , Y, Z).

Output Output is a single integer x, where $x <= 32$. x is the smallest base for which the expression s1 $+$ s2 $=$ s3 is true. If no answer exists, print "No solution".

Sample Input Two separate sample cases:

```
5 3 10

5 3 7
```

Sample Output The corresponding sample output:

```
8

No solution
```

3 Postfix

Task Postfix expressions are arithmetical expressions where the operators come after anything they operate on. Postfix is important because it maintains precedence without the use of brackets.

An example of a postfix expression would be:

3 5 *

This would be evaluated as 3 * 5 resulting in an answer of 15.

6 4 * 2 +

This would be evaluated as 2 + (6 * 4) resulting in an answer of 22.

Input Input will contain a postfix expression which will be no more than 80 characters in length followed by a newline and will contain one expression to be evaluated. All expressions will be correct postfix expressions. There will be one space between each number/operator. The only operators we're interested in is + (addition), - (subtraction), and * (multiplication). There is no division. All numbers are non-negative integers.

Output Your program is to output the result of evaluting the expression. Note that while the input numbers will not be negative, the answer may be negative.

Sample Input Two separate sample cases:

```
3 5 *
```

```
6 4 * 2 +
```

Sample Output The corresponding sample output:

```
15
```

```
26
```

4 Knights

Task In chess, a knight (the horsehead piece) moves differently than any other piece. It moves in an L-type shape. That is, it can move two squares in one direction, then one square in the perpendicular direction (knights can not move diagonally). As an example, consider this:

```
. . . . . . . .
. . . . . . . .
. . . * . * . .
. . * . . . * .
. . . . K . . .
. . * . . . * .
. . . * . * . .
. . . . . . . .
```

The knight is in the square marked by a 'K'. The '*' around it are the possible positions the knight can move on the next turn. The '.' are empty squares.

If you place two knights on the board, things get a little more complicated. We say that a knight can not be taken if there is not a knight that can land on it's square on the next turn. This condition has to hold regardless of what order the knights move (ie, the knights can't 'move' out of the way).

Input Two integers, M, N, separated by exactly one space. The chessboard has size M x M. We're interested in placing N knights on the board such that no knight can take any other knight. The number of knights will be in the range 1 to 36 (inclusive). M will be in the range 1 to 6 (inclusive).

Output Your program should print out the number of ways that we can place N knights on an M x M chessboard. The answer will always fit in an unsigned, 32 bit integer.

Sample Input Two separate sample cases:

```
2 2
```

```
6 4
```

Sample Output The corresponding sample output:

```
6
```

```
26133
```

5 Frame Stacking

Task Consider the following 3 frames placed on an 5x5 grid.

```
MMMMM  NNN..  .....
M...M  N.N..  .....
M...M  NNN..  ..PPP
M...M  .....  ..P.P
MMMMM  .....  ..PPP
```

If the frames are placed top of one another starting with M at the bottom, N in the middle and ending with P on top, any part of a frame covers another it hides that part of the frame below.

```
NNNMM
N.N.M
NNPPP
M.P.P
MMPPP
```

The problem is to determine the order in which the frames are stacked from bottom to top given the stacked frames, given the following constraints:

- The width of the frame is always exactly 1 character and the sides are never shorter than 3 characters.
- It is possible to see at least one part of each of the four sides of a frame. A corner shows two sides.
- The frames will be lettered with capital letters, and no two frames will be assigned the same letter.
- There will be a unique solution to each stack.

Input The first line of input contains two integers, the height (h) and width (w) of the grid, separated by a space. The maximum size of the grid will be 30x30. This is followed by h lines of length w. Each character of each line will be a capital letter representing a visible part of a frame, or a full stop if no frame covers that part of the grid.

Output A single line, containing the letters of the frames in the order they were stacked from bottom to top.

Sample Input Two separate sample cases:

```
9 8
.CCC....
ECBCBB..
DCBCDB..
DCCC.B..
D.B.ABAA
D.BBBB.A
DDDDAD.A
E...AAAA
EEEEEE..

5 5
NNNMM
N.N.M
NNPPP
M.P.P
MMPPP
```

Sample Output The corresponding sample output:

```
EDABC

MNP
```

6 Jugs

Task You have two jugs, A and B, and an infinite supply of water. There are three types of actions that you can use: (1) you can fill a jug, (2) you can empty a jug, and (3) you can pour from one jug to the other. Pouring from one jug to the other stops when the first jug is empty or the second jug is full, whichever comes first.

For example, if A contains 5 litres of water and B contains 6 litres of water and has a capacity of 8, then pouring from A to B leaves B full and 3 litres in A.

A problem is given by a triple (Ca,Cb,N), where Ca and Cb are the capacities of the jugs A and B, respectively, and N is the goal. A solution is a sequence of steps that leaves exactly N litres of water in jug B. The possible steps are

```
fill A
fill B
empty A
empty B
pour A B
pour B A
success
```

where "pour A B" means "pour the contents of jug A into jug B", and "success" means that the goal has been accomplished.

You may assume that the input you are given does have a solution.

Input A single line of three positive integers Ca, Cb, and N, separated by spaces. Ca and Cb are the capacities of jugs A and B, and N is the goal. You can assume $0 < \text{Ca} <= \text{Cb}$ and $\text{N} <= \text{Cb}$ and that A and B are relatively prime to one another.

Output Output from your program will consist of a series of instructions from the list of the potential output lines which will result in jug B containing exactly N litres of water. The last line of output should be the line “success”.

Sample Input Two separate sample cases:

```
3 5 4

5 7 3
```

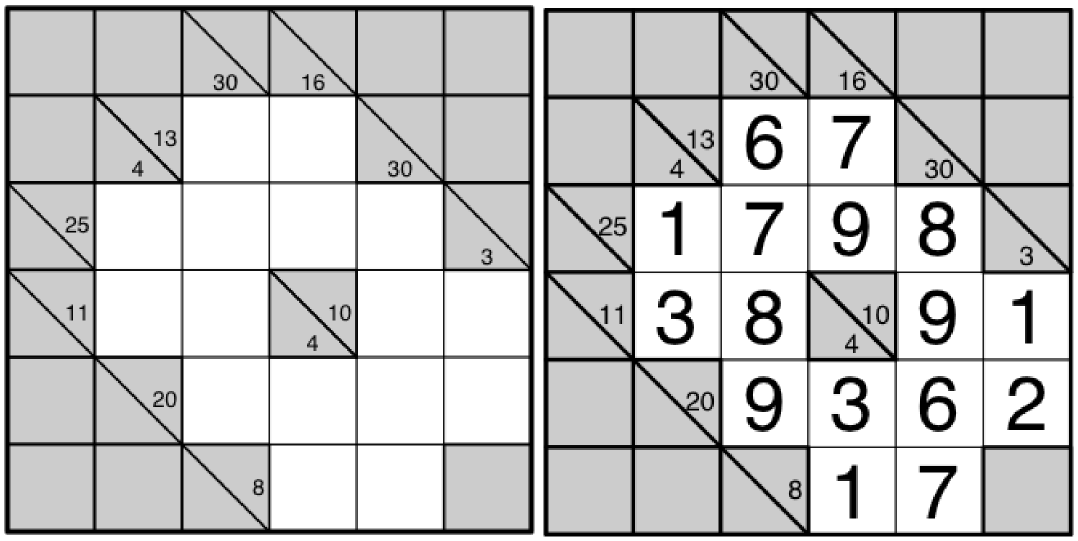
Sample Output The corresponding sample output:

```
fill B
pour B A
empty A
pour B A
fill B
pour B A
success

fill A
pour A B
fill A
pour A B
empty B
pour A B
success
```

7 Kakuro

Task Kakuro, or Cross Sums, is a puzzle played on a grid of filled and empty cells, "black" and "white" respectively. The grid is divided into “entries” - lines of white cells - by the black cells. Some black cells contain a diagonal slash from upper-left to lower-right and a number in one or both halves, such that each horizontal entry has a number in the black half-cell to its immediate left and each vertical entry has a number in the black half-cell immediately above it. These numbers, borrowing crossword terminology, are commonly called “clue”.



(a) Unsolved (b) Solved

Figure 1: A Kakuro puzzle.

The object of the puzzle is to insert a digit from 1 to 9 inclusive into each white cell such that the sum of the numbers in each entry matches the clue associated with it and that no digit is duplicated in any entry. Kakuro puzzles must have a unique solution.

Input The first line of input contains two integers, the height (h) and width (w) of the grid, separated by a space. The maximum size of the grid is 16x16. The height and width of the grid are not required to be equal. Cells are are numbered from 0 to $(h \times w) - 1$, counting from left to right, and top to bottom.

The second line is a list integers separated by spaces, representing numbers of the black cells, excluding "clue" cells, in ascending order. Every line that follows contain five integers separated by spaces, begining with the number of a "clue" cell, followed by the clue for the entry below the cell, the length of the entry below the cell, the clue for the entry to the right of the cell and the length of the entry to the right of the cell. If there is no entry below or to the right of the cell, it is represented by the clue 0 and length 0.

Here is the puzzle from Fig. 1 in this format:

```
6 6
0 1 4 5 6 11 24 30 31 35
2 30 4 0 0
3 16 2 0 0
7 4 2 13 2
10 30 4 0 0
12 0 0 25 4
17 3 2 0 0
18 0 0 11 2
21 4 2 10 2
25 0 0 20 4
32 0 0 8 2
```

Output Your output should be h lines of w characters, representing the solved puzzle grid. For black cells, including "clue" cells, print a full stop. For white cells print the value for that cell in the solved puzzle. The solved puzzle from Fig. 1 would be output like this:

```
......
..67..
.1798.
.38.91
..9362
...17.
```

Sample Input Two separate sample cases:

```
5 5
0 3 4 8 9 13 14 18 19 20 23 24
1 7 3 0 0
2 13 4 0 0
5 0 0 4 2
10 0 0 3 2
15 0 0 6 2
21 0 0 7 1

6 6
0 1 4 5 6 11 24 30 31 35
2 30 4 0 0
3 16 2 0 0
7 4 2 13 2
10 30 4 0 0
12 0 0 25 4
17 3 2 0 0
18 0 0 11 2
21 4 2 10 2
25 0 0 20 4
32 0 0 8 2
```

Sample Output The corresponding sample output:

```
.....
.13..
.21..
.42..
..7..

......
..67..
.1798.
.38.91
..9362
...17.
```

8 Problem of Apollonius

Task In Euclidean plane geometry, Apollonius' problem is to construct circles that are tangent to three given circles in a plane. Two circles are tangental (only intersecting at one point) are called internally tangent if the areas they enclose intersect, and externally tangent if there is no intersection between the areas they enclose. In Fig. 2 the green circle is externally tangent to the three black circles and the red circle is externally tangent to the three black circles.

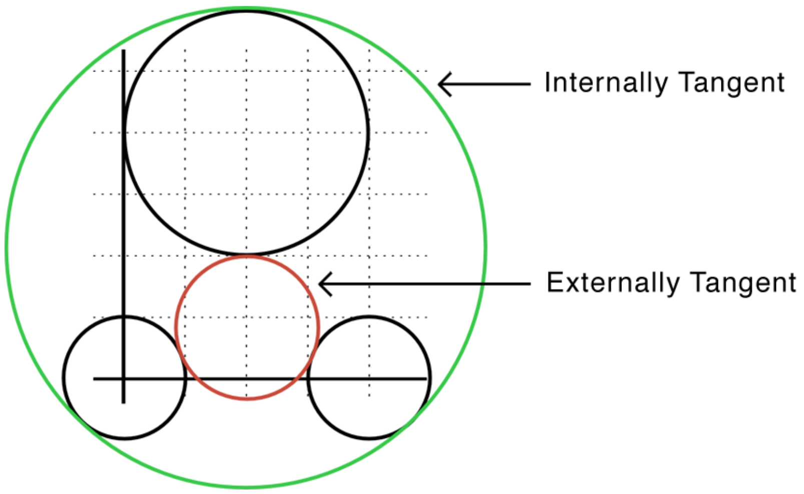


Figure 2: Externally and internally tangent circles.

The aim of this problem is find the circle that is externally tangent to three given circle, or to determine that no such circle exists.

Input Input is 3 lines, each containing 3 integers in the range -100 to 100 inclusive, separated by spaces. Each line describes a circle, the first two integers are the x and y coordinates of the centre of the circle, and the third digit is the radius of the circle.

Output Output is 3 floating point numbers, rounded to 2 decimal places, and separated by spaces, followed by a newline. The line describes a circle that is externally tangent to the 3 input circles.The first two numbers are the x and y coordinates of the centre of the circle, and the third number is the radius of the circle.

If no externally tangent circle exists, output the word 'none' followed by a newline.

Sample Input Two separate sample cases:

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

0 0 1
0 1 1
1 0 1
```

Sample Output The corresponding sample output:

```
2.50 1.27 0.80

none
```