RMC Montana High School Programming Contest

Morning Session, April 26, 2014

Downton Processional

In upper class England, the rules of precedence were complicated. Every time that people were invited to a table to dine, it was very important that the more important people were seated before the less important people. Unfortunately, the rules were complicated enough that you couldn't just list them out. Rather, as an example, individual rules could tell you that person A came before C and person B came before C. Under those rules ABC and BAC were both acceptable orderings.

Your program will be given how many pairs of precedence there are, and then a list of pairs of precedence. Next you'll be given the number of people in the parade order followed by the proposed parade order. Each "person" will be represented by a single capital letter. Your program will the print out "follows protocol" if the parade is consistent with protocol, and "doesn't follow protocol" if it doesn't.

Sample input:

```
4
A C
B C
D B
E A
6
E
A
D
F
B
C
```

Sample output:

```
follows protocol
```

Sample input:

```
2
A C
B C
3
A
C
B
```

Sample output:

```
doesn’t follow protocol
```

FOIL

Your program will multiply out a series of polynomials. Each polynomial will be a polynomial in x (i.e. it can contain terms for x^2 and x^6 but not y^2). Each factor in the data provided will be a single digit (though the final answer could contain a polynomial with a power larger than 9). The constant in front of each term (if any) will be a positive single-digit integer. There will be no spaces and the powers will be written using the ^ notation. Your program will print out the product expanded out, with the terms ordered from biggest to smallest (constant terms are order 0).

Sample input 1:

```
(x+1)(2x^3+x^2)
```

Sample output 1:

```
2x^4+3x^3+x^2
```

Sample input 2:

```
(5+x^5)(x^5+5)
```

Sample output 2:

```
x^10+10x^5+25
```

Sample input 3:

```
(x^2+2x+3)(3x+5)(x^2+x+7)
```

Sample output 3:

```
3x^5+14x^4+51x^3+111x^2+148x+105
```

Repeated numbers

When the number is 64,253 is multiplied by 365 the product is 23,452,345. Notice that the first four digits are the same as the last four digits (2345 and 2345). Write a program that will find any and all integers that can be multiplied by 365 to produce an eight-digit product where the first four digits are the same as the last four digits.

In the example above, no digits were repeated, but in your program, the product can have repetition. For example, 44,884,488. (Hint: Use mod) Also note, that the number 10001 should not be printed out as an answer because it does not have 8 digits. It's also not divisble by 365, so it really, really shouldn't be printed out.

Sample input: *there is no sample input*
Sample output: *that would be giving you the answer ;-)*

Coinworthy

One property of the coinage used by the United States is that each coin is larger than the sum of all the coins with smaller denominations. For example, the dime is worth more than a nickel plus a penny. This is actually true for almost all coin systems worldwide because it has some desirable properties. Write a program to determine if a particular set of coin denominations has this property.

Your program will be given the number of coins, n, followed by n positive integer values representing the value of each coin. The order of the coins is not guaranteed to be in a sorted order. Your program is to sort the list of coins and then determine if the set is worthy to be minted. A list of coins is worthy to be minted if the value of any coin is strictly larger than the sum of all smaller valued coins.

Sample Input 1:
```
4 1 10 5 25
```
Sample Output:
```
1 5 10 25
coinworthy
```

Sample Input 2:
```
3 3 2 1
```
Sample Output 2:
```
1 2 3
not coinworthy
```

Z

Your program is to use a number of print statements to draw a large capital letter ‘Z’ in asterisk characters. All three bars of the ‘Z’ must be the same length. The input to your program will be an integer specifying the length of the bars. The length is guaranteed to be between 3 and 20 inclusive.

Sample Input 1:

```
3
```

Sample Output 1:

```
***
 *
***
```

Sample Input 2:

```
4
```

Sample Output 2:

```
****
  *
 *
****
```

Leaking Plumbing?

One of the basic truths of plumbing systems is that no matter how complicated they get, the volume of liquid flowing into any given joint must equal the volume flowing out it, and for each length of pipe, the volume flowing into one side equals the volume flowing out of it. If either rule is violated then there must be a leak in the system somewhere in the system.

You must write a program to detect if there are leaks in such a network. Your program will have the network of plumbing specified one pipe at a time. For each pipe you'll be given the id of the "starting" joint that it's connected to, the "ending" joint that it's connected to, and the volume of water flowing from the start to the end that is measured somewhere in the length of pipe. This number is guaranteed to be positive, so there will always be a flow *from* the starting joint *to* the ending joint.

The first line of input will be the number of pipes in the system. It is followed by one line of data for each length of pipe in the system. Each line will have the starting joint ID, the ending joint ID and the (integer) volume of water flowing from the start to the finish. Joint IDs will be integers from 1 to the number of joints in the system. Each joint ID must touch at least two pipes, and there will never be a joint ID that is "skipped", i.e. if there is a pipe that refers to joint ID 5, then will be at least one other pipe that touches ID 5 and there will be other pipes in the problem that refer to joint IDs 1-4. The flow value for any given pipe is guaranteed to be an integer in the range of -100 to 100. After analyzing the network, your program needs to either print "leaks" or "doesn't leak."

Sample input 1:
```
3
1 2 1
2 3 1
3 1 1
```
Sample output 1:
```
doesn’t leak
```

Sample input 2:
```
4
1 2 80
2 3 70
3 4 70
4 1 70
```
Sample output 2:
```
leaks
```

Dot to Dot

One of the earliest number puzzles that most of remember is the classic dot-to-dot game: a student is presented with a paper that has a bunch of numbered dots on it, and the student must connect point 1 to 2, 2 to 3, 3 to 4 and so on until all the adjacent numbers are connected. This problem is a variation on the classic dot-to-dot puzzle.

In your version of the puzzle, your program will be told how many coordinates are in the puzzle and then will be presented with a series of (x,y) integer coordinates on the plane. Each coordinate will have positive integer values and will never have an x or y value larger than 50. Each consecutive pair of coordinates is guaranteed to be either be vertically aligned or horizontally aligned. The first and last pair of coordinates are also connected. Your program must then print out the “dot-to-dot” picture, using character art. Any “cell” that is on the border of the resulting shape must be drawn with a *. All other “cells” must be drawn with a period character or a space. The picture must be drawn so that the coordinate (0,0) is in the bottom left of the output.

Sample input

```
6
0 0
0 10
5 10
5 5
15 5
15 0
```

Sample output

```
******
*    *
*    *
*    *
*    *
*    ***********
*              *
*              *
*              *
*              *
****************
```

Pascal’s Triangle

Your program is to display the first N rows of Pascal's Triangle. The program will be given N, where N is a positive integer less than 12, then print the triangle so that its peak is approximately centered over the rest of the output. Spacing must take into account the possibility of numbers of varying sizes. Leave at least two spaces between adjacent numbers in the same row.

Sample Input:

```
6
```

Sample Output:

```
              1
           1    1
         1    2    1
      1    3    3    1
   1     4    6    4    1
1     5    10   10   5     1
```