RMC Western Montana High School Programming Contest

Morning Session, March 15, 2014

Golomb Ruler

In mathematics, a Golomb ruler is a set of marks at integer positions along an imaginary ruler such that no two pairs of marks are the same distance apart. The number of marks on the ruler is its order, and the largest distance between two of its marks is its length. Translation and reflection of a Golomb ruler are considered trivial, so the smallest mark is customarily put at 0 and the next mark at the smaller of its two possible values. Your job is to write a program that determines if a set of marks is a Golomb ruler.

The ruler 0,1,3,7 is a Golomb ruler because it has the following distances:
1-0 = 1, 3-1 = 2, 3-0 = 3, 7-3 = 4, 7-1 = 6, 7-0 = 7
But the ruler 0, 2, 4, 7 is not a Golomb ruler because the distance from 2 to 0 is the same as the distance from 4 to 2.

Your input will be an integer saying the number of marks, followed by an integer for each mark showing the distance from the left edge of the ruler to that mark. Marks will be in increasing order, starting with 0 and ending with the length of the ruler. Your program will either output “this is a Golomb ruler” (if it is), or it will output “Not Golomb” followed by the two pairs of marks that are the same distance apart.

Input dataset example:
4 0 1 3 7
Output example:
This is a Golomb ruler

Input dataset example:
4 0 2 4 7
Output example:
Not Golomb: 0, 2 and 4, 2.

Y not?

Your program is to use a number of print statements to draw a large capital letter ‘Y’ in asterisk characters. All three “legs” of the Y will be the same length. Your program will be input a single positive integer n, which will be the number of stars in each leg of the ‘Y’. n is guaranteed to be between 3 and 20 inclusive.

Sample Input 1:
5

Sample Output 1:

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

Sample Input 2:
4

Sample Output 2:

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

Slightly Magic Square

A magic square is a n by n grid of the integers from 1 to n^2 such that the sum of each row of numbers, each column of numbers, and each diagonal are equal to each other. Your program is designed to help people who are trying to discover magic squares.

Input: your program will be given a positive integer n (which is guaranteed to be less than 10) which will be the size of the square that you're testing. You'll then be given n^2 integers, one row at a time, that are the entries to be put into the square.

Output: your program will output contents of the row, column or diagonal that sums to the largest value. If there is more than one row, column or corner-to-corner diagonal that is maximal, you can choose to print any of them that are maximal. When printing a row, column, or diagonal, it must be printed in order, though that order doesn't matter (i.e. it can be from left to right, right to left, top to bottom, bottom to top, whatever as long as you don't skip around).

Example input 1:

```
3
1 5 9
2 8 3
7 7 7
```

Example output 1:

```
the maximal row, column, or diagonal is 7, 8, 9 which sums to 24.
```

Bi Bim Bap

Write a program that prints the numbers from 0 to n, inclusive. For multiples of two print the string “Bi” instead of the number, for multiples of three print “Bim” instead of the number, and for multiples of five print “Bap”. For numbers which are multiples of both three and five print “Bim Bap”. For multiples of two and five print “Bi Bap”. For multiples of two and three “Bi Bim”. For multiples of two, three, and five print “Yum Yum Yum”. Finally print a blank line followed by a count of the number of words in each category using the following format:

```
Bi: (# of Bi words) Bim: (# of Bim words) Bap: (# of Bap words)
Bi Bim: (# of Bi Bim words) Bim Bap: (# of Bim Bap words) Bi Bap: (# of Bi Bap words)
Yum Yum Yum: (# of Yum Yum Yum words)
```

A number which is a Bi Bim number should only be counted in the Bi Bim count for the final output. It should **not** be counted three times, once in each of the Bi, Bim, and Bi Bim counts. The sum of the final counts in the output should never be greater than n.

The first line of input contains the the positive integer n.

Example input:

```
31
```

Example output: (To conserve space, this output has been formatted in columns)

```
Yum Yum Yum
1
Bi
Bim
Bi
Bap
Bi Bim
7
Bi
Bim
Bi Bap
11
Bi Bim
13
Bi
Bim Bap
Bi
17
Bi Bim
19
Bi Bap
Bim
Bi
23
Bi Bim
Bap
Bi
Bim
Bi
29
Yum Yum Yum
31

Bi: 8 Bim: 4 Bap: 2
Bi Bim: 4 Bim Bap: 1 Bi Bap: 2
Yum Yum Yum: 2
```

Calendar Math

The planet Radnelac takes exactly 157 of its days to circle its sun so its year is 157 days long. Its calendar is further broken up into five months. The first month is Artemis which is 31 days long. The second month is Beatrix which is 32 days long. The third month is Calyx which is 31 days long. The fourth month is Dimelix which is 32 days long. And the last month is Enoughalready which is 31 days long.

On Radnelac (using the Radnelacian calendar), today is Calyx 15, 2014. Your program needs to compute the exact date (using the Radnelacian calendar) of any day that's up to 10000 days into the future from now.

Sample input 1:
157
Sample output 1:
Calyx 15, 2015

Sample input 2:
474
Sample output 2:
Calyx 18, 2017

RackO

You are given a sequence of 6 integers and a spare integer. You are not allowed to change the order of any of the original 6 integers, but you are allowed to insert the “spare” anywhere in that sequence between two of the original integers. The job of your program is to figure out where the “best” place is to insert your spare number, where “best” is defined as the location that makes the sequence contain the longest set of consecutive integers possible.

On the first line of input is the sequence of six integers that can not be changed. Each integer is separated by a single space. On the second line of input is the “spare” integer that is to be inserted into the first sequence of integers.

The first line of output should be the portion of the new sequence of integers that is increasing. The second line of output should be the number of integers in that sequence.

Example input:
111 456 1089 8943 2987 10876
348

Example output:
111 348 456 1089 8943
5

Example input:
100 7 5 90000 88888 77777
100000

Example output:
5 90000 100000
3

Elbonian Credit Cards

Most forms of identification include some sort of error checking protocol that keeps helps reduce problems with people incorrectly entering ID numbers, or problems with card readers not quite getting all the data right. As an example, most credit cards in the United States have 16 digits, but the 16th digit is actually a mathematical formula of the first 15 digits.

Credit cards in Elbonia also use an error correction scheme, although the credit card numbers are only 10 digits (shorter than in the US since it's a smaller country) and the scheme is different (since the Elbonian government has a tight monetary policy and they don't want their citizens to go spending their Elbonian cash in other countries).

Actually, only the first nine digits in an Elboanian Credit Card (ECC) are used to identify the card. The tenth character serves as a check digit to verify that the preceding 9 digits are correctly formed. This check digit is selected so that the value computed as shown in the following algorithm is evenly divisible by 11, a special symbol was selected by the ECC designers to represent 10, and that is the role played by E.

The algorithm used to check an ECC is relatively simple. Two sums, s1 and s2, are computed over the digits of the ECC, with s2 being the sum of the partial sums in s1 after each digit of the ECC is added to it. The ECC is correct if the final value of s2 is evenly divisible by 11.

An example will clarify the procedure. Consider the (correct) ECC 0-13-162959-E. First look at the calculation of s1

digits in the ECC	0	1	3	1	6	2	9	5	9	E(10)
partial sums	0	1	4	5	11	13	22	27	36	46

The calculation of s2 is done by computing the total of the partial sums in the calculation of s1

s2 (running totals)	0	1	5	10	21	34	56	83	119	165

Since 165 is, indeed, evenly divisible by 11, we can confirm that the ECC is valid.

The input of the data for this problem will be one candidate ECC, perhaps preceded and/or followed by additional spaces. No line will contain more than 80 characters, but the candidate ECC may contain illegal characters, and more or fewer than the required 10 digits. Valid ECCs include hyphens at arbitrary locations. The output should either be "valid" or "invalid".

Example input 1:	`0-13-162959E`	Example output 1: `valid`	
Example input 2:	`013162959E`	Example output 2: `valid`	Note: hyphens are irrelevant
Example input 3:	`0-13-1629599`	Example output 3: `invalid`	Note: not multiple of 11
Example input 4:	`0P13Q2959E`	Example output 4: `invalid`	Note: P illegal character
Example input 5:	`0-13-E629599`	Example output 5: `invalid`	Note: E only allowed at last digit