2020 CCSC Eastern Conference Programming Competition

October 24th, 2020

Hood College, Frederick, Maryland

Problem 1 — Dog years

How old are you and your friends in dog years? Let's find out! One human year is equal to seven dog years.

Your program should read lines of input consisting of a name and an integer age in human years. For each input name and human age, the program should print a message of the form

name`, you are` *age* `in dog years`

where *name* is the input name and *age* is the input age converted to dog years. You can assume that the name will not contain any spaces, and that at least one space character will separate the name and the human age. As a special case, if an input line consists of the text `quit`, then the program should exit immediately. You can assume that the overall input will always be terminated by a `quit` line.

Example input:

```
Mariana 20
Emily 22
Pankaj 18
Hiroshi 24
Ming 23
quit
```

Example output (corresponding to the input shown above):

```
Mariana, you are 140 in dog years
Emily, you are 154 in dog years
Pankaj, you are 126 in dog years
Hiroshi, you are 168 in dog years
Ming, you are 161 in dog years
```

Problem 2 — Barn door

A "barn door" can be drawn from text characters as follows:

Width 3

```
+-+
|X|
+-+
```

Width 5

```
+---+
|\ /|
| X |
|/ \|
+---+
```

Width 7

```
+-----+
|\   /|
| \ / |
|  X  |
| / \ |
|/   \|
+-----+
```

Write a program that generates barn door figures of arbitrary widths. Each line of input will specify an integer value. For all of the odd inputs that are 3 or greater, the program should print the barn door figure of that width. For all non-negative even inputs, the program should print a single line with a message of the form

N `is even`

where *N* is the input value. If the input is 1, the program should print a single line with the message

```
Too small
```

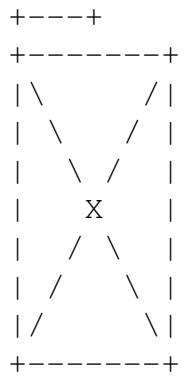
As a special case, if the input value is less than 0, the program should exit immediately. You can assume that the input will be terminated by a line with a negative input value.

Example input:

```
5
9
2
0
1
8
3
-1
```

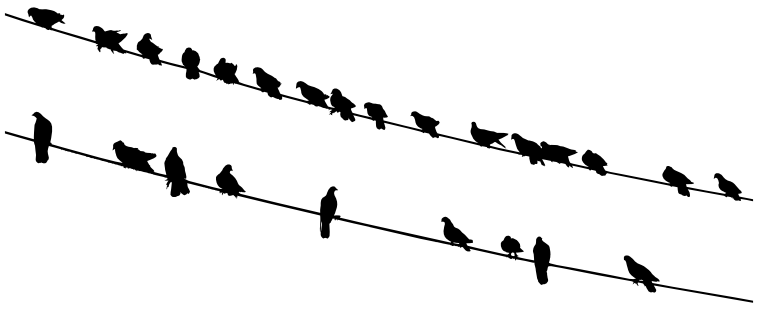
Example output (corresponding to the example input above):

```
+---+
|\ /|
| X |
|/ \|
```




```
2 is even
0 is even
Too small
8 is even
+-+
|X|
+-+
```

Problem 3 — Birbs



Bird watching is one of the most popular hobbies in the Kingdom of Splotvia. In particular, tallying birds perched on overhead wires is a favorite pastime. Even more particularly, when two rows of the fabled Splotvian plaid-throated wren appear on parallel wires, the opportunity for a special tabulation ritual occurs. Each plaid-throated wren has its weight in grams emblazoned on its feathers[1]. Keen Splotvian bird-watchers will note the weight of the *larger* wren in each pair (first pair, second pair, third pair, etc.), and then compute the sum. If the sum is divisible by 7, then it is an auspicious time for a traditional Splotvian eel roast[2]. If the number of birds on each line is unequal, then the excess wrens in the longer line of wrens are ignored.

To aid in tallying, you decide to write a program to tally bird weights as described above, and determine whether the sum is divisible by 7. The program input is pairs of lines, each line consisting of a sequence of one or more integers. For each pair, the program should compute the sum of the larger values in each pair (ignoring excess values on the line containing more values), a print a single line of output of the form

sum `Eels!`

if the sum is divisible by 7, or an output line of the form

sum `No eels :(`

if the sum is not divisible by 7.

As a special case, if a line containing the text "`quit`" is read, then the program should exit immediately without producing any further output.

[Problem continues on next page.]

[1] For some reason, this is always an integer, go figure.

[2] A traditional gathering where famous folk traditions such as the Splotvian squirrel dance can be observed.

Example input:

```
69 7 6 98 4 84 56 88 83 55
48 17 92 61 28 69 51
80 50 78
78 77 88 9 49 73
quit
```

Example output (corresponding to the input shown above):

```
444 No eels :(
245 Eels!
```

Problem 4 — Secret messages

As a discerning reader of the news, you're aware that there is often more to news stories than meets the eye. For example, most people would look at the headline "Two walls contain droll city mural" and see nothing but a mundane story about how a local artist has created a humorous public artwork downtown. You, however, can see the real message: "Tw**o** **w**a**ll**s **cont**ain d**roll** **ci**ty mur**a**l", or, more plainly, "Owls Control CIA".

To make your decoding of secret messages in news stories more efficient, you decide to write a program that checks whether a phrase contains a specific embedded secret message. Specifically, it checks to see whether the phrase has all of the letters in the secret message, in the correct order, ignoring capitalization, spaces, and punctuation.

The input to the program consists of consecutive pairs of lines, the first containing the phrase, and the second containing the secret message. For each such pair, the program should print a single line of output containing "`yes`" or "`no`" depending on whether or not the secret message is embedded in the phrase. As a special case, if the phrase is "`quit`", then the program should exit immediately without reading additional input or producing additional output.

Example input:

```
Two walls contain droll city mural
Owls Control CIA
Two walls contain droll city mural
Elvis shot JFK
Two walls contain droll city mural
Wow, all coin rural
quit
```

Example output (corresponding to the input shown above):

```
yes
no
yes
```

Problem 5 — Word chains

A *word chain* is a sequence of words such that the each word (other than the first) differs from the previous word by only a single letter. More specifically, one word can be transformed into the other by changing a single letter, inserting a single letter, or deleting a single letter.

For example, if the set of words is `man plan cat canal team pan`, then the following word chains are possible (shown one per line):

```
man pan
man pan plan
pan man
pan plan
plan pan
plan pan man
```

Each input to the program is a single line consisting of a positive integer n followed by a set of words. For each input, the program should print a corresponding output listing all word chains of length n or greater that can be formed from the input's words. The resulting word chains should be printed one per line, and should be ordered lexicographically by word, with lexicographical comparisons of words. After all of the discovered word chains have been printed, an additional line consisting of the text "`Done`" should be printed. As a special case, if the input line consists of just "`quit`", then the program should exit immediately without producing further output.

Example input:

```
3 plane team tale tall
2 plane team tale tall
5 cat can tan tam tame mat meat
quit
```

Example output (corresponding to the example input above):

```
Done
tale tall
tall tale
Done
cat can tan tam tame
mat cat can tan tam
mat cat can tan tam tame
meat mat cat can tan
meat mat cat can tan tam
meat mat cat can tan tam tame
```

```
tam tan can cat mat
tam tan can cat mat meat
tame tam tan can cat
tame tam tan can cat mat
tame tam tan can cat mat meat
tan can cat mat meat
Done
```

Problem 6 — Weak assignments

A *weak assignment* has the form

variable = *expression*

where the *expression* defines the value of the variable. The expression may refer to other variables, may have one or more literal integer values, and may contain up to two (2) occurrences of the operators +, −, ⋆, and /. The ⋆ and / operators have a higher precedence than + and −. Operators having the same precedence should be evaluated from left to right. All values are integer values, and the / operator should discard the remainder. (You may assume that the operands to which / is applied will never be negative, and that there will never be an attempt to divide by 0.)

Your program will read lines of input, each specifying one or more weak assignments. Each line will also designate a variable to be evaluated. Input lines have the form

variable <− *assignments*

where *variable* is the name of the variable to be evaluated, and *assignments* is one or more weak assignments, separated by semicolon characters ("`;`"). Each variable name will consist of 1–5 lower case letters. Literal integer values will consist of 1 or more digit characters, optionally preceded by a minus sign ("−"). The individual tokens of the input line will be separated by 1 or more space characters. As a special case, if an input line contains "`quit`", the program should exit immediately without producing further output.

If the evaluation completes successfully, the program should print a single line of text with the final value of the evaluated variable. If the value of the variable cannot be determined, the program should print a single line of text with the text "`Invalid`". Two reasons that the evaluation might fail are:

1. There is an undefined variable
2. There is a circular dependency involving a weak variable whose value is needed in the evaluation

Note that if a circular dependency or undefined variable involves variable(s) that are not necessary for the overall evaluation, it should not prevent the evaluation from succeeding.

Example input:

```
b <- b = a + 3 ; a = 54 / 3
foo <- d = 9 + 3 * a ; baz = d - 4 ; foo = baz - 1 ; a = 9
b <- b = 2 * c ; c = b - 4 ; a = 11
a <- b = 2 * c ; c = b - 4 ; a = 11
y <- y = z - 4 ; z = x * 9
z <- z = 42 ; q = a + 10
quit
```

Example output (corresponding to the example input above):

```
21
31
Invalid
11
Invalid
42
```

Problem 7 — We Get the Widgets

A factory contains two conveyer belts, L and R. L and R each process strings of parts named A through Z into widgets to sell on the totally legit website Banazom.com. The parts from L and R will be connected but each piece can only be connected to similar parts, meaning A connects only to A, B to B, etc. The parts can be rotated such that they may be connected. A rotation puts the first part at the end of the widget. (e.g. BOX rotates to OXB). We would like to arrange the parts such that each pair of parts will create the Absolute Best Widget. The Absolute Best Widget has the largest number of connected parts at the front.

Input: The first line in the input contains the number of test cases to be run, N. The remaining lines contain pairs of parts for L and R one on each line, N times. Only capital letters are used.

Output: For each line (for each L and R) print the smallest number of rotations needed to create the Absolute Best Widget. Print -1 if there are no matching parts in L and R.

Example input:

```
4
HELLO
BELL
WORLD
RLDWO
PARTYTIME
MEARTECARTI
XYZ
ABC
```

Example output (corresponding to the example input above):

```
1
1
0
3
5
9
-1
-1
```

Problem 8 — Fail First Search

Oh no! PROBOT 21,000 has a glitch and we don't have time to fix it. We're going to have to work around it. PROBOT was designed to seek and destroy annoying little huma—I mean monsters but our programmers went on strike before they finished debugging. (Apparently, they think we should pay them?). PROBOT would be given a map of monsters[3] in a neighborhood. It would use the map to locate its targets and deal with them peacefully[4]. Every minute of travel time takes up gas, so PROBOT must keep track of how much time it will take to complete its task. Unfortunately, PROBOT does not seem to make the right decisions. It will only travel to houses that take an even number of minutes to travel and only travels to the furthest house away. (Luckily, it never visits the same house twice and if it has nowhere else to go, it just turns off). Find out how much time PROBOT will waste on each in each of these neighborhoods before it turns off.

Input: Each neighborhood's homes are connected by sidewalks. The neighborhood input is in the format (u, v, w) where u and v are houses connected by a sidewalk that takes w minutes to reach. No two sidewalks take the same time to travel. The houses u and v are named as integers between 0 and 14 with PROBOT always beginning at 0. The first line in the input contains the number of neighborhoods, and the first line for each neighborhood contains the number of sidewalks available. Note that PROBOT's poor mobile system only allows it to travel on sidewalks though it can thankfully travel in either direction.

Output: For each neighborhood, output the number of minutes PROBOT will waste.

Example input:

```
2
6
0 1 7
0 4 6
1 2 1
1 3 10
2 3 2
3 4 4
1
0 1 9
```

Example output (corresponding to the example input above):

```
20
0
```

[3]Definitely the truth

[4]Also completely true

Problem 9 — Spinlatin

HELLLOOEO, fellow kids! Welcome to your first Spinlatin class! Spinlatin is the raddest new language and all the on fleek cats are using it. It's just GROOVY, or should I say GROOOVYVYRVY so let's get spinning!

To translate a word from English into Spinlatin, you just have to follow these simple rules. Every two adjacent letters in the original word form a pair (a, b). Translate that word one letter at a time, and when any letter is added to the new word, replace it with the (a, b) pair if it exists. If multiple (a, b) pairs exist for a letter, add them in alphabetical order. Each letter may only be used up to 3 times.

Input: The first line is the number of words to translate, N. The remaining N lines are single words in all capital letters A – Z.

Output: Translate each word into Spinlatin.

Example input:

```
5
PI
ABC
EYES
HELLO
ALFALFA
```

Example output (corresponding to the example input above):

```
PII
ABCBCC
ESYESYEYS
HELLLOOEO
ALFALFALF
```

Problem 10 — Primogenerator

Smeagol loves pretty, precious trinkets. We don't have any jewelry, but we can make pretty numbers. A palprime is a prime number that is also a palindrome. It's the same number forward and backward and can only be divided by itself and 1. Can it get any more precious?

Input: The first line represents N, the number of test cases. Each other line contains a single integer between 2 and 10,000,000.

Output: For each case, print "Yes" if those numbers can be rearranged into a palprime number. Otherwise, print "No".

Example input:

```
4
7
511
992
123412
```

Example output (corresponding to the example input above):

```
Yes
Yes
Yes
No
```

Problem 11 — Garden design

Traditional Splotvian gardens are always square, and consist of flowers planted in a grid pattern. Because a Splotvian garden always has room for a gnome statue in the center, a square region in the middle of the garden is always left empty. Gnome statues vary in size, so the size of the empty square van vary.

One absolute rule of Splotvian garden design is that when placing a flower, the flowers directly above, below, left, or right must not have the same color.

In order to know how many options you have in designing a Splotvian garden, you decide to write a program. Each input to the program is on a single line, and consists of three integer values:

- The number of distinct flower colors
- The number of rows/columns of the overall garden
- The number of rows/columns of the empty square in the middle of the garden

For example, if the input is `3 5 3`, then the garden will have 3 distinct flower colors, the overall garden is 5x5, and the empty square in the middle is 3x3. Note that Splotvian gardens must be symmetrical in shape, so if the outer dimension is odd, the inner (empty) dimension must also be odd, and if the outer dimension is even, the inner (empty) dimension must be even. As a special case, if an input line contains just the text "`quit`", the program should exit immediately without producing any further output.

For each input, the program prints the number of distinct garden designs that are possible. For example, if the input is `3 3 1`, *one possible* design is the following (with the digits `1`, `2`, and `3` indicating one of the three possible flower colors):

```
323
2 1
312
```

However, this is one of 258 possible designs for this input.

As special case, if the input is invalid, then the program should print a line with the text "`Invalid`". A valid input must be one where all three values are positive, the second and third values are either both even or both odd, and the second value is at least 2 greater than the third value.

Note that the number of possible gardens can be very large. You may assume that your program will be allowed a "reasonable" amount of running time to enumerate the possible garden configurations.

Example input:

```
3 3 1
3 2 1
2 4 2
```

```
3 4 2
3 5 3
3 6 4
4 3 1
quit
```

Example output (corresponding to the example input above):

```
258
Invalid
2
4098
65538
1048578
6564
```