Robotique
CRC
Robotics

CRC

PROGRAMMING CHALLENGE

BLOCK B

PROBLEM BOOKLET

A FEW NOTES

- The complete rules are in section 5 of the rulebook
- This challenge was made for small teams of programmers, we do not recommend having more than 4 players on your team
- The all-around bonus applies only if you complete one/two problem(s) in every category of BOTH block A AND block B
- You will NOT be able to retry these problems during the 2nd day (block B) of the competition, so plan your strategy accordingly
- Anything submitted after the time limit will not be counted
- The judges should be able to simply copy paste the inputs all in ONE bunch and receive the outputs all in ONE bunch for correction all in the console
- Program smart: making an input reader should only be done once and then tweaked for other problems
- Your team is not expected to be able to complete everything easily so don't be shocked if you see a large amount of problems
- You are also free to come ask us questions on the problems!

STRUCTURE

Each problem contains a small introduction like this about the basics of the problem and what is required to solve it. Before every problem, in the title, will appear both its category and its index within that category. The number in parentheses corresponds to the number of points given for a successful solve.

Input and output specification:

In these two sections, we specify what the inputs will be and what form they will take, and we also say what outputs are required for the code to produce and in what format they shall be.

Sample input and output:

In these two sections, you will find a sample input (that often has multiple entries itself) in the sample input and what your program should produce for such an input in the sample output.

Explanation of the first output:

Sometimes, the problem might still be hard to understand after those sections, which is why there will also be a usually brief explanation of the logic that was used to reach the first output from the first input.

Table of Contents

NEUTRON STAR (NS)

NS1: RGBs everywhere (20)

For an obscure reason your personal automated assistant isn't able anymore to read the hexadecimal formatting. This format is used to have a compact notation for colors. Since your assistant can't read that format anymore you need to convert a string with the hex formatting for colors. The format is #XXYYZZ in which XX is the red value, YY the green value and ZZ the blue value. Those values are in hexadecimal (base 16) so are between 0 (00) and 255 (FF).

Input specification:

The first line contains an integer *n* that corresponds to the number of entries needing to be transformed. Every one of the *n* lines below contains a string of the hexadecimal color value in the format #XXYYZZ which needs to be converted.

Output specification:

For each entry of hex value you need to give the RGB values of the color in the following format: (R,G,B).

Sample input:

```
4
#3AC267
#E2495A
#91347F
#FF0000
```

Sample output:

```
(58,194,103)
(226,73,90)
(145,52,127)
(255,0,0)
```

Explanation of the first output:

For the first example we ignore the #, then we take 3A in hex which is 58 (3*16 + 10). Then we take the two next characters C2 that give 194 (12*16 + 2) and then the last one 67 is 103 in decimal (6*16 + 7).

NS2: Caesar, Jules (35)

We have found old stone tablets in a crater on an abandoned planet. The tablets contain strange letters that seem to be repeating. Maybe those symbols could be numbers instead of words that would work pretty much like the roman numeral system. We can automatically translate those symbols to the roman numeral system to make the counting easier. You now need to translate those roman numerals into decimal numbers.

We will use M=1000, D=500, C=100, L=50, X=10, V=5 and I=1 just like the roman numeral system. As a reminder, 3 and 8 are written as III and VIII but 4 and 9 are written as IV and IX. The order of the symbols has its importance. The same rule applies for 40, 90, 400 and 900.

Input specification:

The first line contains an integer *n* that corresponds to the number of entries needing to be transformed. Every one of the *n* lines below contains the string that is a roman numeral to convert.

Output specification:

For every roman numeral in the input you need to give the decimal number.

Sample input:

```
4
DCCCXLII
MMDIV
CLXIX
MMCDLV
```

Sample output:

```
842
2504
1519
2455
```

Explanation of the first output:

D is 500, each C is 100 and because there's 3 of them we add them up to a partial total of 800. Then we have X (10) followed by L (50) so we need to subtract 10 from 50 to get 40. Adding to that we have 2 I. If we had all of the parts together we get 800 + 40 + 2 = 842.

NS3: Exoplanets classification II (50)

It's great to calculate the surface area of a planet but there is much more to consider! We can't just get onto a big boulder because it's spacious. In this problem we will obviously only consider exoplanets. We will still need to calculate to surface of the planet that we have to live on with this formula:

$$A_P = 4\pi R_P^2$$

This time , we will consider that the **COMPLETE** area of the planet is suited for humans. But, we will add two other criterias to guide the planet selection. The first added criteria is true or false to know whether the planet has an atmosphere. The other one is number based criteria for fresh water on the surface of the planet. A planet passes the freshwater threshold if it has at least 30 million of km^2 of its surface covered by freshwater. First we need to consider if there is an atmosphere on the planet (1). Then we consider if there is enough freshwater on the planet (2). Finally our last criteria is a decreasing ranking of the surface area. The exoplanets need to be sorted so the most compatible is first.

Input specification:

The first line contains an integer *n* that corresponds to the number of entries needing to be transformed. Every one of the *n* lines below contains the name of the planet, its radius (in km), the percentage covered by freshwater (between 0 and 100) and finally OUI if the planet has an atmosphere or NON if it doesn't.

Output specification:

You will need to sort the planet by the explained criterias. On each line we need to see the name of the planet followed by "ATMOSPHERE" if there is one and then "FRESHWATER" if there is enough on the exoplanet. If the criteria for freshwater or atmosphere is not met, "FRESHWATER" or "ATMOSPHERE" should not be written. Then at the end we need the surface of the planet. The sorting of the planet should be made by sections which are ATMOSPHERE FRESHWATER -> ATMOSPHERE -> FRESHWATER -> (planets with neither). In each section the planets need to be ordered by the most surface area to the least surface area.

Sample input:

```
6
Galaxnar 2660 3 OUI
Ictus 9240 2.3 NON
Amogus 11244 5 OUI
Agecanonix 9756 11.8 NON
Moothonroo 5700 0 OUI
Daxor 3875 1.7 NON
```

Sample output:

```
Amogus ATMOSPHERE FRESHWATER  1588735273
Moothonroo ATMOSPHERE 408281381
Galaxnar ATMOSPHERE 88914611
Agecanonix FRESHWATER 1196061324
Ictus 1072886564
Daxor 188691909
```

Explanation of the output:

The planet Galaxnar for example has an atmosphere but not enough freshwater which puts it behind Amogus that has both and in front of Agecanonix which only has freshwater. In its section with Moothonroo the other planet has more space available for the humans so it gets ranked higher.

NS4: *Caesar cycle intensifies* (60)

After you've found out how to rearrange letters in a word by cycling them it is now time to decode phrases or even better, messages. As a reminder to offset a word you need to cycle all the letters to the left by the offset. As the letters get to the leftmost side of the message it needs to cycle back to the right end of the message. To decode the message you won't be given an offset value but rather a keyword to find in your message. This keyword of m letters will indicate the good offset when it will perfectly fit in a m letter word.

You can then decode the message based on the two following principles. First off, the key word is contained exactly one time and fits in only one place in the message. This should be able to find the right offset to decode the message. For the spaces and the ponctuation, it should not move with the cycling of the letters. Only the letters need to change place from the given encrypted message to the decrypted one.

Input specification:

The first line contains an integer *n* that corresponds to the number of entries needing to be transformed. Every one of the *n* lines below contains the keyword to decode the message. Note: the keynote and the message mais contain accents. The keyword will be separated from the rest of the message by a ";"

Output specification:

For each message to decode, you will need to output the number of letters of the offset going to the left that get to the message for the first time. Then after a ";" you will need to give the complete message decrypted.

Sample input:

```
2
contrat;su r sama is onilafa it uncontratd ere pa ration.
tragically;oneatethel, astdoug hnu ttr agic allysome.
```

Sample output:

```
11;il a fait un contrat de réparation sur sa maison.
21;tragically, someone ate the last doughnut.
```

Explanation of the first output:

We need to cycle the message 11 positions to the left to get the keyword "contrat" when you have the offset you can offset the whole message to the left by 11 and then print out the result.

NS5: Shortcuts only (65)

In your trip you go by a few planets and you want to make sure to choose the fastest path possible. You need to find the shortest path between all the planets. The first planet given in the input will be the starting point and the last one the endpoint. The other planets given in between need to be ordered to give the shortest possible path. The distance between two planets can be found with this formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Input specification:

The first line contains an integer *n* that corresponds to the number of entries needing to be transformed. Every one of the *n* lines below contains the coordinates of the planets in 3 dimensions. There will be a maximum of 8 planets. All the coordinates will be separated by a ";"

Output specification:

For each entry of sets of planets you will need to give the planets (by the position in the entry) organized to have the smallest total distance. After the planets numbers you need to indicate the total travel distance rounded to the nearest integer.

Sample input:

```
2
3.5;6,8;12.4 90.5;65.2;6.0 73.5;63.4;1.8 9.4;45.2;46.1 23.3;54.0;28.9
3.5;6.8;12.4 9.4;45.2;46.1 90.5;65.2;6.0
```

Sample output:

```
1 4 2 3 5 220
1 2 3 144
```

Explanation of the first output:

We start from planet 1 of coordinates 3.5;6,8;12.4 and to find the shortest total path we go to planet 4 followed by planet 2 and then 3 before going to our final destination, planet 5. The total distance for all the planets is 219.507056840963 rounded to 220.

SUPERMASSIVE BLACK HOLE (SBH)

SBH1: Primo de facto (45)

After having tested which numbers were prime, we are now asking you to decompose numbers into their prime factors in order to study the properties of those primes' powers. We remind you of the fundamental arithmetic theorem which states that all whole numbers bigger than 1 is either a prime number or can be expressed as a multiplication of primes.
For example:

$$1035 = 3 \times 3 \times 5 \times 23 = 3^2 \times 5 \times 23$$

You will therefore need to give the prime factor decomposition (knowing that 1 is not prime) with the factors being ordered from smallest to biggest (without taking exponents into account).

Input specification:

The first line contains an integer *n* that corresponds to the number of entries that you will need to treat. Each of the *n* lines under contains a number between 2 and 10 000 that needs to be decomposed into its prime factors.

Output specification:

For each given number, your program will need to determine its prime factors and present them in ascending order. The factors' powers will be written as X^Y where X is the factor and Y is the exponent. If the exponent is 1, only write X. The multiplications will be represented by "x", and all operations and factors will be separated by spaces.

Sample input:

```
4
65
32
867
3892
```

Sample output:

```
5 x 13
2^5
3 x 17^2
2^2 x 7 x 139
```

Explanation of the first output:

By testing the prime factors you find that for 65 you have 5 and 13 as prime factors. So you put 5 * 13 as the final output.

SBH2: Polynormal (50)

In certain applications, a polynomial will be more useful under its developed form. By the fundamental theorem of algebra, every polynomial can be written as its roots (values of x for which the equation equals 0) and possesses as many roots as its degree. For example, in the polynomial of degree 2:

$$ax^2 + bx + c = (x - x_1)(x - x_2)$$

where a, b and c are constants, x_1 x_2 are the polynomial's roots.

For this problem, you can assume that **all roots are different**. You will therefore need to find as many roots as the degree of the polynomial. You also know that **all roots are whole numbers between -30 and 30 inclusively**. The process of factorisation therefore amounts to finding all the roots, then writing the polynomial under its factored form. We remind you that the degree of a polynomial corresponds to the highest exponent of x in it.

Input specification:

The first line contains an integer n that corresponds to the number of entries you will need to treat. Each of the n lines under contain a polynomial of equal or decreasing power. It is entirely possible that a power of x be missing in the polynomial (which would be equivalent to that power's coefficient being 0). For example, a polynomial of degree 3 could have no term at x^2. The terms will be formatted as follows: $a_i x^i$ where i is an integer representing the exponent and a_i is the coefficient of the entire term (there is an implied multiplication between the a_i and x^i). Every term will be separated by either a "+" or "-". **The exponents of *x* in a polynomial are all different from one another**. All operators and terms will be separated by spaces.

Output specification:

You will need to give every polynomial under its factored form **by ordering its roots from smallest to biggest** with the following format: $(x-x_1)(x-x_2)$... etc. If the root is negative, you cannot write two "-" in a row. For example, if the root is -3, you will write (x+3) rather than (x--3).

Sample input:

```
3
x^4 + 20x^3 - 87x^2 - 806x + 3080
x^2 - 169
x^3 - 46x^2 + 553x - 1740
```

Sample output:

```
(x+22)(x+7)(x-4)(x-5)
(x+13)(x-13)
(x-5)(x-12)(x-29)
```

Explanation of the first output:

The roots of the polynomial are -22, -7, 4 and 5. By ordering them, we get the given factorisation.

SBH3: Duality (75)

After the horror of complex numbers last year, we decided to tone down the challenge. This year your calculator of dual numbers needs to be fixed. You now need to implement a new one for adding, subtracting and multiplication.

A dual number can be written in a $a+b\varepsilon$ format in which a and b are constants and ε can be seen as a variable that follows these rules: $\varepsilon \neq 0$ mais $\varepsilon^2=0$. This may seem unusual but can be interpreted as the b number is so small that we can estimate its squared value to 0. But when the power isn't even we still need to consider the number. The three available operations looks like this:

$$(a + b\varepsilon) + (c + d\varepsilon) = (a + c) + (b + d)\varepsilon$$
$$(a + b\varepsilon) - (c + d\varepsilon) = (a - c) + (b - d)\varepsilon$$
$$(a + b\varepsilon) \times (c + d\varepsilon) = ac + (bc + ad)\varepsilon + bd\varepsilon^2$$

where $bd\varepsilon^2$ takes the value 0 because of the dual numbers properties. You are required to have a functional calculator for adding, subtracting and multiplicating.

Input specification:

The first line contains an integer *n* that corresponds to the number of entries that you will need to treat. Each of the *n* lines under contains a string of operations that need to be done from left to right. Adding, subtracting and multiplying will be represented by these symbols "+", "-" et "*". The dual numbers are in parentheses with the format ($a+b\varepsilon$) with a and b non zero numbers but can be either positive or negative integers. Note that if the b part of the dual number is lesser than 0 then it will be presented as ($a-3\varepsilon$). In that case a can also be a negative number.

Output specification:

For each entry you will need to give the resulting dual number. Contrary to the input, the output may have no dual part and only be a regular number. In that case 0+3ε will become 3ε. The same thing applies if the number is negative.

Sample input:

```
3
(1+2ε)+(5-3ε)-(-2-8ε)
(7-7ε)*(1+2ε)*(3-5ε)
(1+2ε)*(1-2ε)-(1+2ε)*(1-2ε)
```

Sample output:

```
8+7ε
21-14ε
-2ε
```

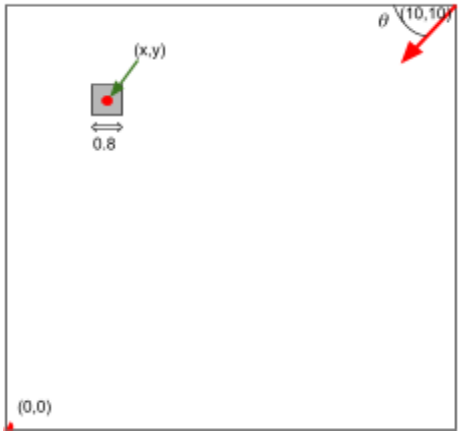
Explanation of the first output:

As it is simply adding and subtraction we add the regular parts together and we add the dual parts with each other. We then get 1 + 5 - - 2 = 8 for the normal part and 2 - 3 - - 8 = 7 for the dual part.

SBH4: Smoke and mirrors (85)

Buckle up because this is going to get a bit long. Basically, we are trying to calibrate a high intensity laser pointer for future experiments. To do so, the pointer will need to pass a security test and your work will be to verify if the laser touches the target and if so, if its intensity is at a dangerous level or not.

To start off, let's talk about the testing room. The testing room measures 10 by 10 units with its origin coordinates at the bottom left corner. The laser pointer is placed in the upper right corner and is oriented to shoot into the room with an angle θ measured from the horizontal line as shown on the graphic:



The equation for the slope of the laser's path is as follows, knowing that the function passes by the (10,10) point:

$$slope = m = \frac{\Delta y}{\Delta x} = \frac{sin\theta}{cos\theta} \quad \text{where } y = mx + b$$

You can find *b* with a point on the linear function and the slope. **Careful! It's possible that the function that you will be using to calculate the *cos* and *sin* will need your angle θ to be in radian or degrees. This will depend on the chosen language and libraries used.** The walls of the room are straight mirrors that reflect the laser with the same leaving angle as the incoming angle. Mathematically, this is equivalent to changing the slope of the laser from *m* to *-m* and finding the new y-intercept given the point of contact. **This rule is not valid for the corners, which your code will need to account for.** If the laser touches a corner perfectly, the equation for the new trajectory is exactly the same and the laser will go back on its tracks.

The target possesses a square *hitbox* around its position that is 0.8 units large, with the target at its center. Even hitting the corner of the *hitbox* counts as a *hit.*

The mirrors will not be perfect and the laser's intensity will decrease by 3% every time it gets reflected. The new intensity is given by the equation:

$$I_f = 0.97 \times I_i$$

where *i* stands for initial and *f* stands for final. **The lasers will no longer be dangerous when their intensity is under 20W.** You will need to make simulations to know if the target will get hit or not.

Input specification:

The first line contains an integer *n* that corresponds to the number of entries that you will need to treat. Each of the *n* lines under contains the position (x,y) of the target with the initial angle θ (in degrees) and the initial intensity (en W) of the laser.

Output specification:

For each entry you will have to say "SECURE" if the laser miss the target as long as it is not dangerous also with the number of bounce to become in a safe range of power. In the other case, if the laser does hit the target, you have to output "HIT" followed by the intensity of the laser at the point of contact (rounded to the millimeter).

Sample input:

```
3
(4.7,5.8) 0 100
(8.3,1.2) 82.1 30
(1.5,4.5) 5.73 40
```

Sample output:

```
SECURE 53
HIT 29.1
SECURE 23

(29.100 is also okay here since the value is exact)
```

Explanation of the first output:

Because the laser is fired fully horizontally and the target is around the middle of the room, there is no way for the target to get hit. The laser goes under 20W intensity after the 53th bounce, so we write 53.