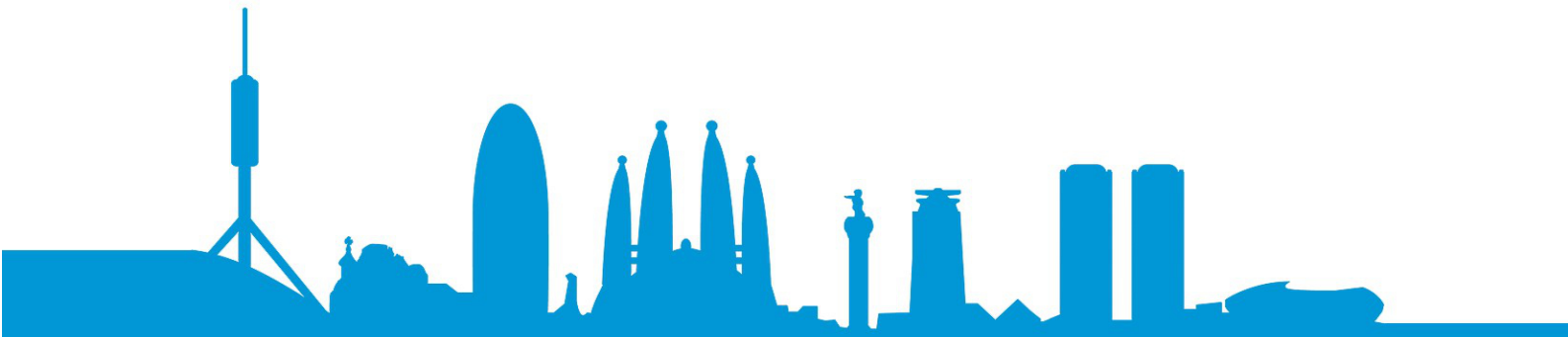


CodeWars 2017
Barcelona

Problems And Solutions

#	Problem	Points	Id
0	Hello jury	1	P89365
1	A simple echo	1	P62572
2	Let's party	2	P82870
3	Run or die	2	P82425
4	Adding consecutives and precedents	3	P91393
5	Filling doughnuts	3	P30349
6	Football pools	3	P70403
7	The arrival	3	P12479
8	Uppercase and lowercase hacking	3	P35410
9	Does my car fit in that parking slot?	3	P17464
10	The right change	4	P81311
11	Alice's adventures in horizontal printers land	4	P41118
12	Bike race	4	P77761
13	The Niffler wants it all	5	P74020
14	The apple fall	5	P70103
15	HP-GL plotter controller	14	P55110
16	Ludic numbers	7	P71690
17	How long is your TV?	7	P92491
18	Fidelity miles converter	7	P87033
19	Stairs architect	7	P23489
20	By car or by plane?	10	P20555
21	The living pyramid	10	P67719
22	Magic squares	10	P26685
23	Roman calculator	10	P69585
24	We live in a spherical world	10	P68859
25	World chess championship	12	P72534
26	Goldbach's conjecture	14	P15225
27	Time is money	14	P99792
28	Shuttle bus service schedule	16	P83537
29	Transmission tower	16	P75076
30	Hero of the Core	18	P99494
31	The CodeWars is MINE!	20	P52663
32	Two-colorable graphs	22	P20140

2 Let´s party

2 points

Introduction

You will spend a weekend with your friends in the Costa Brava and you want to bring some music in a pen drive of certain gigabytes (GB). Considering that a song is approximately 11 megabytes (MB). You need to calculate the number of songs that fit on your pen drive.

Notice that there are 1024 MB in 1 GB.

Input

The input will be a single integer number indicating the capacity of the drive in GB.

```
1
```

Output

Print out the number of songs that will fit on the specified drive:

```
93 songs
```

Solution

```
#include <stdint.h>
#include <iostream>

int main()
{

    uint32_t penDrvSize = 0;

    std::cin >> penDrvSize;

    std::cout << ((penDrvSize * 1024) / 11) << " songs" << std::endl;

}
```

3 Run or die

2 points

Introduction

The running world organization is asking us to create a tool to calculate how many stops they need to create in a giving contest to avoid any dehydration problem on the participants. In order to calculate this number they are going to provide the number of kilometers of a giving running contest and the number of kilometers before a human being can suffer dehydration. The result we need to provide is the number of stops needed.

Input

The first line of the input will be the number of the total kilometers of the running contest. The second line of the input is the number of kilometers before dehydration.

Example 1

```
100
10
```

Example 2

```
25
6
```

Output

The output should be the number of stops (as an integer value) in the form:

Example 1

```
10
```

Example 2

```
4
```

Solution

```
#include <stdint.h>
#include <iostream>

int main()
{

    uint32_t totalKms = 0;
    uint32_t numKmsBeforeDehydration = 0;

    std::cin >> totalKms;

    std::cin >> numKmsBeforeDehydration;

    if (numKmsBeforeDehydration == 0)
    {
        std::cout << "Division by zero" << std::endl;
    }
    else
    {
        std::cout << (totalKms / numKmsBeforeDehydration) << std::endl;
    }

}
```

4 Adding consecutives and precedents

3 points

Introduction

Given a pair of numbers n and m, both values are integers but m is greater than 0, find out the sum of the m consecutives numbers to n and the sum of the m precedents numbers to n.

Example:
n = 1
m = 5
Sum m consecutives to n: 2 + 3 + 4 + 5 + 6 = 20
Sum m precedents to n: 0 + (-1) + (-2) + (-3) + (-4) = -10

Input

The input will be a pair of values expressed as integers representing ***n*** and ***m*** respectively.
1
5

Output

The program must output two numbers. The first one is the sum of the m consecutive numbers to n and the second is the sum of the m precedent numbers to n.
`20 -10`

Solution

```
#include <stdint.h>
#include <iostream>

int main()
{

    uint32_t n = 0, m = 0;
    int32_t consec = 0, prec = 0;

    std::cin >> n;
    std::cin >> m;


    for (uint8_t i = 1; i <= m; i++)
    {
       consec += n+i;
       prec += n-i;
    }

    std::cout << consec << " " << prec << std::endl;

}
```

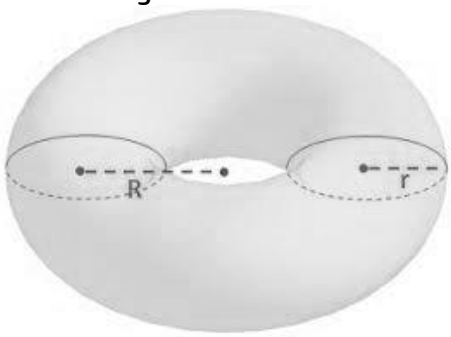
5 Filling doughnuts

3 points

Introduction

Your new job as a bakery assistant presents a new challenge. The baker has created a new doughnut filled with banana cream. You must provide him a program to calculate the exact volume of cream needed per doughnut.

The interior volume of a torus is close to a doughnut and its volume can be calculated with this formula:



$$V = 2\pi^2 R r^2$$

Where R is the distance from the center of the tube to the center of the torus and r is the radius of the tube. Consider the π value as 3.14.

HINT: Considering that you are going to play with real numbers use as much precision as you can.

Input

The input will be a pair of values expressed as a positive integers representing R and r in millimeters respectively.

```
38
16
```

Output

The program must output the volume of cream in cubic millimeters with a resolution of 2 decimals.

```
191828.38 cubic millimeters are needed
```

Solution

```
#include <stdint.h>
#include <math.h>
#include <iostream>
#include <iomanip>

int main()
{

    uint32_t r = 0, R = 0;
    double V = 0;

    const double Pi = 3.14;

    std::cin >> R;
    std::cin >> r;

    V = 2.0*pow(Pi,2)*R*pow(r,2);

    std::cout << std::fixed << std::setprecision(2) << V << " cubic millimeters are needed" << std::endl;
}
```

6 Football pools

3 points

Introduction

The “Quiniela” is the name of a Spanish gaming, managed by the National Lottery, which is based on the National Football League Championship.

The bet is placed on a list of 15 matches, normally 10 of First Division and 5 of Second Division. The possible values for each of the matches are 1 (meaning that the local team wins), X (meaning tie) or 2 (meaning that the visiting team wins).

You have been requested to develop a system to evaluate the number of hits of every bet.

Input

The input will be two strings. The first string contains the 15 results of the matches while the second string represents a single bet that must be evaluated.

X111XXX21X21222
11XXX2XXX221111

Output

The program must output the number of hits for current bet.
5

Solution

```
#include <stdint.h>
#include <iostream>

int main()
{

    uint32_t hits = 0;
    std::string results, bet;

    std::cin >> results;
    std::cin >> bet;

    for (uint8_t i=0; i < results.length(); i++)
    {
        if (results[i] == bet[i])
        {
            hits++;
        }
    }

    std::cout << hits << std::endl;

}
```

7 The arrival

3 points

Introduction

The YETI (Yet searching for Extra Terrestrial Intelligence) is receiving a signal from a remote solar system. The message is a long sequence containing characters and numbers and as you may imagine, it seems impossible to decipher...
But you have been asked to isolate the signal by splitting it into two sequences -- one with characters and another with numbers – in order to continue the investigation. Let's do it!

Input

The input will be a string containing characters and numbers with no spaces.

```
0HELLO1WORLD2!3WE4CAME5FROM6A7GALAXY8FAR9FAR0AWAY
```

Output

The program must output two strings. The first one contains all the characters except numbers and the second contains only numbers.

```
HELLOWORLD!WECAMEFROMAGALAXYFARFARAWAY
01234567890
```

Solution

```
#include <stdint.h>
#include <iostream>

int main()
{

    std::string inputMessage;
    std::string everyThingExceptNumbersMessage;
    std::string numbersMessage;

    std::cin >> inputMessage;

    for (uint8_t i=0; i < inputMessage.length(); i++)
    {
        if (inputMessage[i] >= '0' && inputMessage[i] <= '9')
        {
          numbersMessage.push_back(inputMessage[i]);
        }
        else
        {
          everyThingExceptNumbersMessage.push_back(inputMessage[i]);
        }
    }

    std::cout << everyThingExceptNumbersMessage << std::endl;
    std::cout << numbersMessage << std::endl;

}
```

8 Uppercase and lowercase hacking

3 points

Introduction

The CodeWars participants' database has been hacked. Somebody thought it would be funny to replace all the uppercase letters with lowercase letters in the names and the reverse. So, when it was time to print the participants' id badges we realized that the typography looked a little crazy. Can you help to restore the content of the database?

Input

The input will be a string containing the hacked name of the participant.

```
jOHN
```

Output

The program must output the "unhacked" name, restored to its original form.

```
John
```

Solution

```
#include <stdint.h>
#include <iostream>

int main()
{

    std::string inputMessage;
    std::string outputMessage;

    getline(std::cin, inputMessage); // Read entire line into s

    for (uint8_t i=0; i <= inputMessage.length(); i++)
    {
        if (isupper(inputMessage[i]))
        {
            outputMessage.push_back(tolower(inputMessage[i]));
        }
        else if (islower(inputMessage[i]))
        {
            outputMessage.push_back(toupper(inputMessage[i]));
        }
        else
        {
            outputMessage.push_back(inputMessage[i]);
        }
    }

    std::cout << outputMessage << std::endl;
}
```

9 Does my car fit in that parking spot?

3 points

Introduction

We are the happy owners of a brand new car. We know how much space we need to park it properly and the car has a sensor that indicates the size of the parking spots it detects.

For instance, if we need 1.765 x 4.329 meters, we will be able to park on a spot that is this size or bigger.

Input

`<Width needed> <Height needed>` in millimeters separated by one space (no decimals).
`<Width found> <Height found>` found in millimeters separated by one space (no decimals).

Example 1

```
1765 4329
1765 4000
```

Example 2

```
4000 2000
2000 4000
```

Output

"`yes`" if the car fits both horizontally and vertically; "`no`" otherwise.

Example 1

```
no
```

Example 2

```
yes
```

Solution

```
import java.util.Scanner;

public class Main {

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);

		int carWidth = sc.nextInt();
		int carHeight = sc.nextInt();

		int spotWidth = sc.nextInt();
		int spotHeight = sc.nextInt();

		if ( (spotWidth>=carWidth  && spotHeight>=carHeight) ||
			 (spotWidth>=carHeight && spotHeight>=carWidth))
			System.out.println("yes");
		else
			System.out.println("no");

		sc.close();
	}

}
```

10 The right change

4 points

Introduction

Until the time when everybody uses proximity cards and smartphones, we can still buy things in cash. Often times, we hand over a banknote whose value exceeds the total amount of the purchased items; the cashier will then hand us back the change in smaller notes or coins.

One traditional way of verifying that the change that we get back is correct is to start counting the change starting from the receipt's amount until we reach the note's value.

For instance, if we buy a book priced at 8.50€ with a 20€ note, the cashier may deliver the change as follows:

1. Here are 0.50€ that make up for 9€.
2. Here are 1.00€ that make up for 10€.
3. Here are 10.00€ that make up for 20€.

and we are good to go with our new book and the right change in our pocket.

This validation involves additions instead of subtractions, which are generally easier for humans.

You are asked to implement a computer program that, in this fashion, validates that the change returned back is correct.

Input

`Book's price` (an amount between 0.01 and 100.00€)
`Banknote` you are paying with (just one). Valid banknotes have the following value in Euros {5.00, 10.00, 20.00, 50.00, 100.00}.
`One line for every change iteration` (maximum 20), where each line includes the amount that the cashier is returning to us as a banknote or coin with any of the following values {0.01,0.02,0.05, 0.10, 0.20, 0.50, 1.00, 2.00, 5.00, 10.00, 20.00, 50.00}.
`0` ("0" in the input indicates that the casher is not giving us any more change)

Example 1

```
8.50
20.00
0.50
1.00
10.00
0
```

Example 2

```
7
10
1.00
0.5
1.00
0
```

Output

Output "`Right`" if the change is correct; output "`Wrong`" if the change is not correct. The order and the number of coins or banknotes returned by the cashier is not relevant as long as it matches the expected change.

Example 1
`Right`

Example 2
`Wrong`

Solution

```
#include <iostream>
#include <cmath>

using namespace std;

int main() {
        int bill, acc;
        float d;
        cin >> d; acc = round(d*100);
        cin >> d; bill = round(d*100);
        while ( (cin >> d)&& d!=0)
        {
                acc += round(d*100);
        }

        cout << ((acc==bill)?"Right":"Wrong") << endl;
        return 0;
}
```

11 Alice's adventures in horizontal printers land

4 points

Introduction

After coming back from her visit to Wonderland, Alice was playing in her room and while facing the looking-glass, she started asking herself how could be the world beyond it ...
She entered a fantastic world by climbing though the mirror, and lived such fantastic adventures, that in some cases it seemed like a dream.

Of course...everything would have been faster with our help, being such fantastic programmers as we are, we could have shown her this mirrored world in a matter of minutes.

Program specification

Your program should take a matrix representing the pixels of an image, and flip it horizontally.

Input

The matrix is an ASCII-based file, and all the rows have the same length. The matrix ends up with a # character (this character is forbidden in the image), and has at least one line.

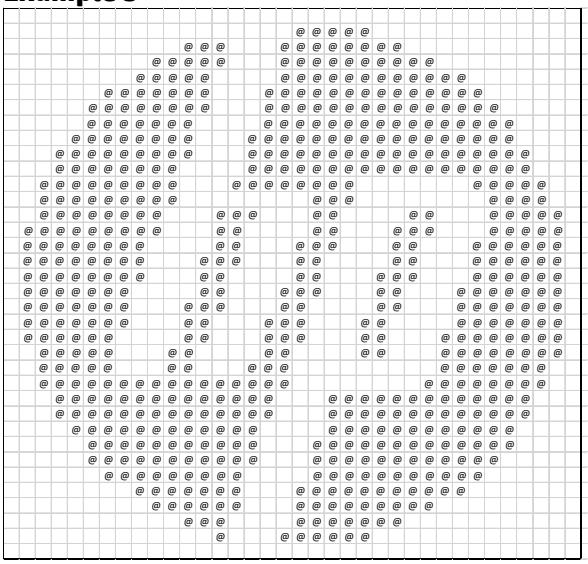
Example 1

```
01234
#
```

Example 2

```
ABC
DEF
#
```

Example 3

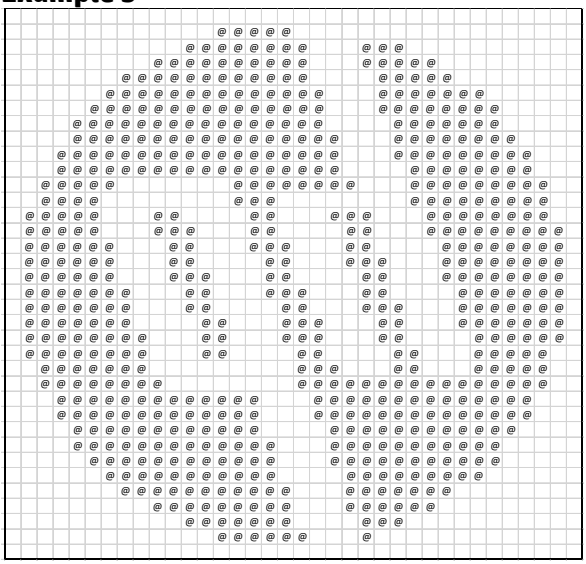


Output

Example 1
43210

Example 2
CBA
FED

Example 3



Solution

```
# Read the input: a ascii matrix with and unknown number or rows/colums containing the immage
# It will read each line, until get #, and print it mirrowed

line=input()
while ( line != '#' ):
    print(line[::-1])
    line=input()
```

12 Bike race

4 points

Introduction

Next month, there will be a bike race in Montsec and the race organizers have asked you to create an application to calculate the final classification. It will be a timed race, the competitors will start at different times, and the important thing is the time they take to complete the circuit.
To know how long a competitor takes, there will be two sensors that will save both the timestamp the cyclist starts and the time he crosses the finish line. This data will be sent to your application.
The timestamps are epoch time, meaning that each timestamp will save the seconds elapsed since 1970.

Input

The input will be a sequence of all the competitors names with their initial and their final timestamp value, ending with a #.

```
Bernard 1483519913 1483523201
Joel 1483520440 1483523498
Lucas 1483521040 1483524254
#
```

Output

The output should be a list of all the competitors, each of them with their final result expressed in seconds, and sorted by first to last position.

```
Joel 3058
Lucas 3214
Bernard 3288
```

Solution

```
#include <iostream>
#include <stdint.h>
#include <map>

int main() {

	std::string name;
	uint32_t timestamp1, timestamp2, time;
	std::multimap<uint32_t, std::string> competitors;

	std::cin >> name;
	while(name != "#")
	{
		std::cin >> timestamp1;
		std::cin >> timestamp2;

		// Discard competitor when timestamp is wrong
		if(timestamp2 > timestamp1)
		{
			time = timestamp2 - timestamp1;
		}
		else
		{
			continue;
		}

		competitors.insert(std::pair<uint32_t, std::string>(time,name));

		std::cin >> name;

	}

	//multimap sort automatically their values by time

	for( std::multimap<uint32_t, std::string>::iterator it = competitors.begin();
		 it != competitors.end(); ++it)
	{
		std::cout << (*it).second << " " << (*it).first << std::endl;
	}

	return 0;
}
```

13 The Niffler wants it all

5 points

Introduction

Newt Scamander's curious monster, the Niffler, loves stealing things, and the more sparkly they are, the more he feels attracted to them. Newt has a mission, he has to save the city of New York, but the Niffler has escaped and he is stealing every jewel, coin or any sparkly thing he sees. But before Newt can fully focus on saving the city, he has to clean the mess and catch the Niffler back again, so you are going to help him to do so.

Newt is chasing the Niffler, and from time to time he finds the Niffler sneaking into a shop and he tries to catch him in the middle of the encounter. You are going to write a program that records the amount of sparkly things the Niffler steals, and the number of those sparkly objects Newt gets back in each encounter, so you can tell him whether he finally catches the Niffler or not. But be aware, the Niffler has a magical pocket where he stores everything he steals, so Newt has to get all objects he did not get in previous encounters.

Input

You will be given sequences of encounters between Newt and the Niffler. Each input line will be an encounter and will contain two values: The first one, the number of objects the Niffler has been able to steal in this particular encounter, and the second value, the number of objects Newt has been able to retrieve from the Niffler in the encounter.
Finally, a -1 (minus one) will indicate the end of the sequences.

•All values will always be integers greater or equal to zero.

•The total number of objects retrieved by Newt will always be less or equal to the number of objects stolen by the Niffler.

•There will always be at least one input sequence before -1.

Example of Input:

```
2 0
10 0
5 10
8 8
0 0
0 7
-1
```

Output

The output will be the result of the persecution for every input line, and a final line indicating whether Newt catches the Niffler.
Different messages have to be displayed for the three possible situations:

•When the Niffler increments its advantage over Newt, that is, he has stolen more objects vs those retrieved by Newt, 'The Niffler is escaping. +DIFFERENCE' has to be displayed.

•When Newt is getting closer, that is, he is retrieving more objects than the Niffler is stealing, then 'Newt is getting closer to the Niffler. -DIFFERENCE'.

•When the number of objects the Niffler steals remains the same as Newt has retrieved, then 'Newt is keeping the distance with the Niffler'.

In the first two cases, DIFFERENCE will be the amount of object the Niffler or Newt have advanced respect to the other.
Finally, if Newt catches the Niffler, that is, Newt has retrieved all objects stolen by the Niffler, the last line will be 'Newt catches the Niffler', otherwise, 'The Niffler has escaped'.

Example of Output:

```
The Niffler is escaping. +2
The Niffler is escaping. +10
Newt is getting closer to the Niffler. -5
Newt is keeping the distance with the Niffler
Newt is keeping the distance with the Niffler
Newt is getting closer to the Niffler. -7
Newt catches the Niffler
```

Solution

```
#include <iostream>
#include <vector>
using namespace std;

typedef unsigned int uint;

int main()
{
    int currentDiff = 0;
    int totalDiff   = 0;
    int aux, aux1;

    cin >> aux;
    while ( aux != -1 )
    {
        cin >> aux1;

        // Result of current encounter
        //
        currentDiff = aux - aux1;

        // Output current status
        //
        if ( currentDiff > 0 )
        {
            cout << "The Niffler is escaping. +" << currentDiff << endl;
        }
        else if ( currentDiff < 0 )
        {
            cout << "Newt is getting closer to the Niffler. " << currentDiff << endl;
        }
        else
        {
            cout << "Newt is keeping the distance with the Niffler" << endl;
        }

        // Accumulate to the total result
        //
        totalDiff += currentDiff;

        cin >> aux;
    }

    // Output final result
    if ( totalDiff == 0 )
    {
        cout << "Newt catches the Niffler" << endl;
    }
    else
    {
        cout << "The Niffler has escaped" << endl;
    }
}
```

14 The apple fall

5 points

Introduction

The Interplanetary Space Organization is studying the gravity in different planets with the most advanced orbital telescope by checking how long it takes for an apple to fall from the top of the apple tree to the ground on each planet.

The data collected by the scientists for each planet consist of the height of the tree and the time it took the apple to land on the ground.

Your mission is to find out which one of the planets in this research has the strongest gravity.

$$h = \frac{1}{2} g t^2$$

Input format:
[Number of planets in this research (N)]
[Height of the tree in meters] [Time to fall in seconds] <-- One line for each planet numbered 1, 2, 3,...,N

Output format:
Planet number [Number of planet] has the most gravity.

Input

```
2
3.25 2
20 2.474
```

Output

```
Planet number 2 has the most gravity.
```

Solution

```
import java.util.Scanner;

public class Main {

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);

		int N = sc.nextInt(); sc.nextLine();

		int highestGravityIndex = 0;
		double highestGravity = 0.0;

		for (int planetIndex=1;planetIndex<=N;planetIndex++){
			double height = sc.nextDouble();
			double time = sc.nextDouble();

			// Compute the acceleration (gravity) in this planet.
			double gravity = (2.0*height)/(time*time);

			if (gravity > highestGravity){
				highestGravityIndex = planetIndex;
				highestGravity = gravity;
				}
		}

		System.out.println("Planet number "+highestGravityIndex+" has the most gravity.");
		sc.close();
	}
}
```

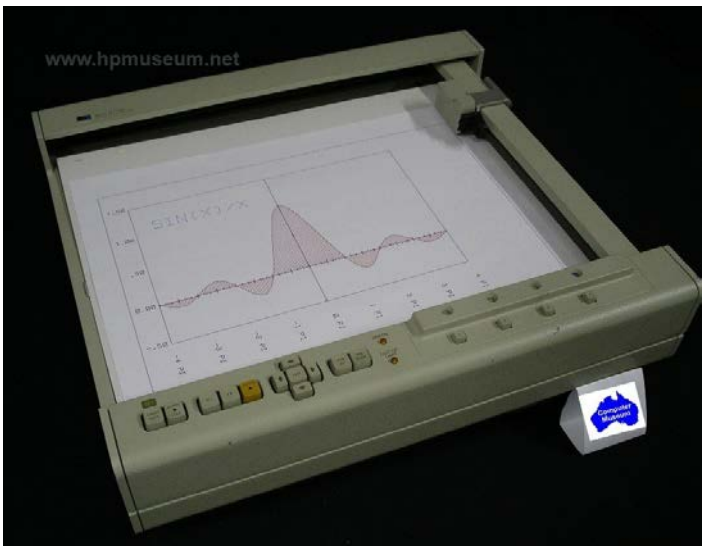
15 HP-GL plotter controller

14 points

Special Award

Introduction

With the launch of the Hewlett Packard 9872A flatbed plotter in 1977, HP began naming its 2-letter mnemonic graphics language as HP-GL (Hewlett Packard Graphics Language). Nowadays, modern printers still implementing HP-GL.



HP-GL is a command set embedded in the ROM of pen plotters to help reduce the work required by applications programmers to create plotted output. HP-GL uses two-letter mnemonics as instructions for drawing lines, circles, text, and simple symbols.

Program specification

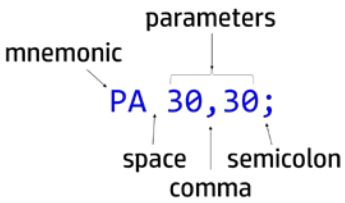
Your program should read an list of simplified HP-GL commands, interpret the commands, and translate them to simple instructions to control an X-Y table.
These output instructions are the movements on the X and Y axis, and a command to raise or lower the pen.

Input: A list of HP-GL commands (one per line), ending with a #
Output: The translated list of commands to the X-Y table (one per line), ending with a #

HP-GL reference guide (reduced)

For simplification, we'll use a reduced set of instructions.
The whole reference guide is not needed for this exercise, but in case of interest, you can find it at:
www.hpmuseum.net/document.php?catfile=213

HP-GL commands have four components: a mnemonic, parameters, separators, and a terminator. Refer to the following illustration of a typical HP-GL command and the description of its components.



Mnemonic--The two-character mnemonic defines the command's function. The mnemonic will be uppercase.
Parameters are used in certain instructions to tell the device to complete the instruction in a particular way.
Separators are used to separate one parameter from the next. They will be a comma.
Terminator separates one instruction from the next. They will be a semicolon.

For this exercise, we'll only consider the next commands:

CO	Comment
PD	Pen Down
PU	Pen Up
PA	Plot Absolute

CO, Comment
CO *["c...c"];*

Allows comments to be inserted within an HP-GL instruction sequence. HP-GL comments are ignored by the device.

PA, Plot Absolute
PA *[X,Y [,X,Y […]]] ;*

Establishes absolute plotting (vs relative plotting) and moves the pen to the specified coordinates.

- No parameters: Establishes absolute plotting for subsequent instructions.
- X,Y coordinates: Specify the absolute location to which the pen moves. When you include more than one coordinate pair, the pen moves to each point in the order given, using the current pen up/down status. If the pen is up, PA moves the pen to the point, if the pen is down, PA draws a line to the point.

PD, Pen Down
PD *[X,Y [,X,Y […]]] ;*

This instruction lowers the pen to draw lines on the page.

- No parameters: Lower the pen.
- X,Y coordinates: Lower the pen and draws to the points specified. You can specify as many X,Y coordinate pairs as you want. When you include more than one coordinate pair, the device draws to each point in the order given.

Example

```
PA    10,10;                Begin absolute plotting from coordinate (10,10).
PD    2500,10,10,1500,10,10;  Set the Pen Down and draw lines between the specified points.
```

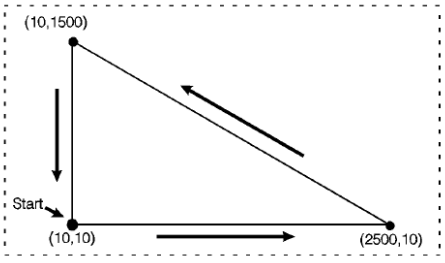


Figure 119. Using the PD (Pen Down) Instruction

PU, Pen Up
PU *[X,Y [,X,Y […]]] ;*

This instruction raises the pen to prevent drawing lines on the page.

- No parameters: Raises the pen.
- X,Y coordinates: Raises the pen and moves to the points specified. You can specify as many X,Y coordinate pairs as you want. When you include more than one coordinate pair, the device moves to each point in the order given.

X-Y table reference guide

Our X-Y table has two steeper motors, for the X and Y axis, and one actuator to lower/raise the pen. Your program must translate from the HP-GL commands, to these ones.
The set of instructions that takes is very simple:

VECTOR GROUP	
MA	Move absolute
PD	Pen Down
PU	Pen Up

MA, Move Absolute

Purpose
MA *X,Y ;*

Each instruction has one, and only one, pair of coordinates.
This instruction moves the pen to the X,Y coordinates using the current pen up/down status. If the pen in up, moves the pen to the point, if the pen is down, draws a line to the point

PD, Pen Down

PD *;*

Lowers the device's pen.

PU, Pen Up

PU *;*

Raises the device's pen.

Example

The input is:

```
CO "Triangle";
PU;
PU;
PA 10,10;
PD 2500,10,10,1500,10,10;
#
```

The output is the list of commands to be send to the X-Y table:

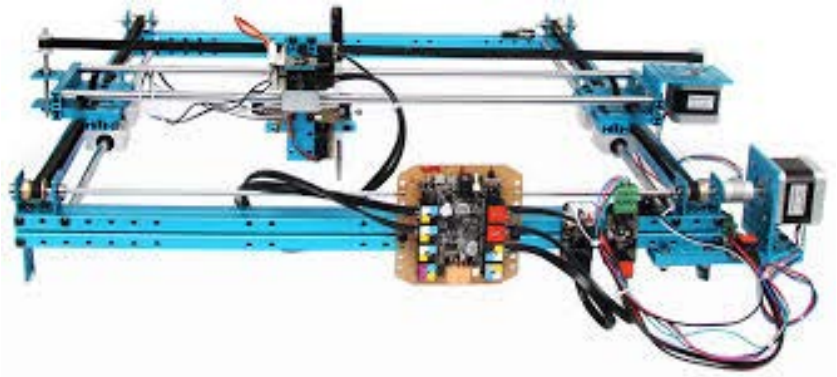
```
PU;
PU;
MA 10,10;
PD;
MA 2500,10;
MA 10,1500;
MA 10,10;
#
```

Surprise output....!!!!

Be aware!!! If you solve this exercise, you will see the connection of your PC's virtual world, with the physical one where we live ;-)

Programming is not just a serial of commands that nerds use to display some game hero in your PC screen: as in old HP's plotters, nowadays, programming is the core of many domestic to industrial machines; the bridge of many devices with our environment.
From the controller of one of the blinds at your home (with end of travel, obstacle detection, timer programming...), to the ABS algorithm of your car, there are many applications where you can find software modules, interacting with physical components.

Each time a team submits a correct solution to the problem, your fresh written code is going to be used to send a plot to the X-Y table with your team id and the order of submittal. Take this print back at home with you !



Do you want to have a closer look to the X-Y table? the code inside? send your own HP-GL file?....feel free to come to the table after the competition.

Solution

```
#include <iostream>
#include <algorithm>
#include <sstream>

using namespace std;

int main() {
	string line, op;
	long x,y;
	while(getline(cin,line) && line!="#")
	{
		// Convert all separators in spaces to facilitate splitting into tokens.
		replace( line.begin(), line.end(), ',', ' ' );
		replace( line.begin(), line.end(), ';', ' ' );
		istringstream ss(line);
		ss >> op;
		if (op=="CO")  // Ignore comments.
			continue;
		if (op=="PU" || op=="PD")
			cout << op <<";"<< endl;
		while (ss >> x >> y)
			cout << "MA " << x << "," << y << ";" << endl;
	}
	cout << "#" <<endl;
	return 0;
}
```

16 Ludic numbers

7 points

Introduction

Ludic numbers are related to prime numbers as they are generated by a sieving process. The first ludic number is 1.

To generate succeeding ludic numbers create an array of increasing integers starting from 2.

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 ...

Take the first member of the resultant array as the next ludic number: 2. Remove every 2nd indexed item from the array (including the first).

~~2~~ 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ 21 ~~22~~ 23 ~~24~~ 25 ~~26~~ ...

Take the first member of the resultant array as the next ludic number 3. Remove every 3rd indexed item from the array (including the first).

~~3~~ 5 7 ~~9~~ 11 13 ~~15~~ 17 19 ~~21~~ 23 25 ~~27~~ 29 31 ~~33~~ 35 37 ~~39~~ 41 43 ~~45~~ 47 49 ~~51~~ ...

Take the first member of the resultant array as the next ludic number 5.Remove every 5th indexed item from the array (including the first).

~~5~~ 7 11 13 17 ~~19~~ 23 25 29 31 ~~35~~ 37 41 43 47 ~~49~~ 53 55 59 61 ~~65~~ 67 71 73 77 ...

Take the first member of the resultant array as the next ludic number 7. Remove every 7th indexed item from the array (including the first).

~~7~~ 11 13 17 23 25 29 ~~31~~ 37 41 43 47 53 55 ~~59~~ 61 67 71 73 77 83 ~~85~~ 89 91 97 ...

Take the first member of the current array as the next ludic number L. Remove every Lth indexed item from the array (including the first).

Write a program that given a number n writes all “ludic numbers” smaller than n.

Input

The input of the program is a positive integer not bigger than 1000.

```
50
```

Output

The program must find all ludic numbers smaller than the provided one.

```
1 2 3 5 7 11 13 17 23 25 29 37 41 43 47
```

Solution

```
import java.util.ArrayList;
import java.util.List;

public class Main {

 public static List < Integer > ludicUpTo(int n) {
  List < Integer > ludics = new ArrayList < Integer > (n);
  for (int i = 1; i <= n; i++) { //fill the initial list
   ludics.add(i);
  }
  //start at index 1 because the first ludic number is 1 and we don't remove anything for it
  for (int cursor = 1; cursor < ludics.size(); cursor++) {
   int thisLudic = ludics.get(cursor); //the first item in the list is a ludic number
   int removeCursor = cursor + thisLudic; //start removing that many items later
   while (removeCursor < ludics.size()) {
    ludics.remove(removeCursor); //remove the next item
    removeCursor = removeCursor + thisLudic - 1; //move the removal cursor up as many spaces as we need to, then back
one to make up for the item we just removed
   }
  }
  return ludics;
 }

 public static void main(String[] args) {
  int max = 0;
  if (args.length > 0) {
   try {
    max = Integer.parseInt(args[0]);
   } catch (NumberFormatException e) {
    System.err.println("Argument" + args[0] + " must be an integer.");
    System.exit(1);
   }
  }

  System.out.println(ludicUpTo(max));
 }
}
```

17 How long is your TV?

7 points

Introduction

You want to buy a TV in and the information they give you in a shop are: inches of the TV-screen (diagonal), aspect ratio of the width and height of the TV and the width of the TV-frame in centimeters. When you arrive at home, you don't know if the TV would fit in your TV-cabinet so you write a program to figure it out.

NOTE: 1 inch = 2.54cm

Input

The input of the program consist in three lines: fist line are your tv inches (always a real number), second line is the aspect ratio (16:9, 4:3, ...) separately by ':' and third line how wide is the edge.
I.e.: The input of a 55" TV with 16:9 aspect ratio and a 1.5cm of edge.

```
55
16:9
1.5
```

Output

```
124.8x71.5cm
```

Solution

```
#include <iostream>
#include <cmath>
#include <string>
#include <cstdio>
#include <vector>
#include <sstream>
#include <iomanip>
#include <stdlib.h>

using namespace std;
// how long is your TV:
// Input: inches + aspect ratio + tv border
// Output: width x heigth in cm
int main()
{
    // READ INPUT STRUCTURE:
    // EX: INCHES  -> number
    //     ASPECT_RATIO -> string like 16:9
    //     TVBORDERINCM -> number

    int inches = 0;
    std::string aspectRatio;
    float borderInCm = 0;

    cin >> inches;
    cin >> aspectRatio;
    cin >> borderInCm;

    std::vector<int> array;
    std::stringstream ss(aspectRatio);
    std::string tmp;
    while(std::getline(ss, tmp, ':'))
    {
        array.push_back(atoi(tmp.c_str()));
    }

    int x = array.at(0);
    int y = array.at(1);

    if (inches <= 0)
    {
        cout << "Inches must be positive" << endl;
        return -1;
    }

    if (x <= 0 || y <= 0)
    {
        cout << "Aspect ratio is not correct" << endl;
        return -1;
    }

    const float inchToCm = 2.54;
    float angleInRad = atan((float)y / (float)x);

    float hypotenuse = inchToCm * inches;

    float height = sin(angleInRad) * hypotenuse + borderInCm*2;
    float width = cos(angleInRad) * hypotenuse + borderInCm*2;

    printf("%.1fx%.1fcm\n", width, height);

    return 0;
}
```

18 Fidelity miles converter

7 points

Introduction

HP Managers use to travel lots of times per year in order to visit the different HP sites and facilities around the world. Airlines reward customer's loyalty by granting points depending on the single integer number of miles traveled, which are converted to points that can be exchanged for tickets at a discounted price, or even free. Usually, the more miles you travel, the bigger reward you can get.

HP has an agreement with its favorite airline, that offers the following points plan:
- You earn one point per each completed mile you have flight.
- You earn 2 points per each completed mile after having flown between 10000 and 25000 miles.
- You earn 3 points per each completed mile after having flown between 25000 and 50000 miles.
- You earn 5 points per each completed mile after having flown more than 50000 miles.

Input

Input will be one or more sequences formed by a name plus one or more numbers. The '#' character represents the end of the input.

```
Jessica
8625.31
3922.91
7543.02
5903.94
1905.24
1827.88
3858.48
2437.70
8625.31
9537.85
Joseph
5903.94
715.06
Sarah
5903.94
715.06
5786.84
2387.21
3349.56
6104.07
1187.34
Brandon
5536.89
5536.89
#
```

Output

Output will be the list of travelers and the points accumulated, ordered from highest to lowest.

```
Jessica 135910
Sarah 41293
Brandon 12144
Joseph 6618
```

Solution

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        HashMap<String,Integer> passengersMilesMap = new HashMap<>();
        HashMap<String,Integer> passengersPointsMap = new HashMap<>();

        // Read Input :
        Scanner s = new Scanner(System.in);
        while (true) {
            String name = s.next();

            if(name.equals("#")) break;

            Integer miles = 0;

            while (s.hasNextFloat()) {
                Float f = s.nextFloat();
                miles = miles + f.intValue();
            }

            passengersMilesMap.put(name,miles);
        }

        // Calculate points:
        for(Map.Entry<String,Integer> entry : passengersMilesMap.entrySet()) {
            Integer miles = entry.getValue();
            Integer points = 0;

            // 2nd tier : (10.000,25.000)
            if (miles >= 10000) {
                points += 10000;                // TIER 1 MAX = 10.000

                // 2nd tier : (25.000,50.000)
                if (miles >= 25000) {
                    points += 15000 * 2;        // TIER 2 MAX = 15.000

                    // 3rd tier : (50.000,+)
                    if (miles > 50000)
                        points += (25000 * 3) + (miles - 50000) * 5; // TIER 3 MAX = 25.000
                    else
                        points += (miles - 25000) * 3;

                } else points += (miles - 10000) * 2;
            } else points += miles;

            passengersPointsMap.put(entry.getKey(), points);
        }

        // Order points:
        while (!passengersPointsMap.isEmpty()) {
            Map.Entry<String,Integer> highestEntry = null;

            for(Map.Entry<String,Integer> entry : passengersPointsMap.entrySet()) {
                if (highestEntry == null) highestEntry = entry;
                else if (highestEntry.getValue() < entry.getValue()) highestEntry = entry;
            }

            System.out.println(highestEntry.getKey() + " " + highestEntry.getValue());
            passengersPointsMap.remove(highestEntry.getKey());
        }
    }
}
```

19 Stairs architect

7 points

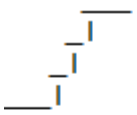
Introduction

There is a stairs architect who wants a new program to show the stairs he designs for his costumers, for agile designing, in order to add value to the business.
Could you help building a program that shows the final design of the stairs?
He only wants to pass the program how many steps are needed.

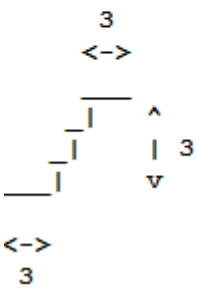
Input

3

Output



Note the output:



Solution

```
# Python 3

def stairs(n):
    if n > 0:
        print(" " * (2 * n - 1) + "_" * n)
        for i in range(n-1):
            print(" " * (2 * n - i - 2) + "_|")
        print("_" * n + "|")


steps = int(input())
stairs(steps)
```

20 By car or by plane?

10 points

Introduction

Easter is near and it's time to plan our vacation.
My friends would like to go to Florence this year and we have not decided yet whether going by car or by plane.
Going by plane costs the ticket price per person, whereas going by car costs the same price (fuel, tolls) for 1 or 5 people.
Also if we go by car we will have to spend a night in a hotel and nor the dinner and the breakfast are included.
The hotel we have found has a fee per room up to 3 people.
It's not clear how many of us will go to Florence and we would like to know what's the cheapest way to go.
Can you help us to decide?

Input

The input starts with the cost of the plane ticket (per person), the cost for a car for up to 5 people, then the cost of the room (up to 3 people), the cost of the dinner and breakfast (per person), and finally the minimum and maximum number of friends that plan to go.

```
200 400 150 20 2 6
```

Output

For every possible number of friends, it must show the cost per person going by plane and by car.
The latest line must show the number of friends whose cost per person is the best, along the transportation media and the total cost.

```
2 friends. Cost per person by plane: 200. Cost per person by car: 295
3 friends. Cost per person by plane: 200. Cost per person by car: 203
4 friends. Cost per person by plane: 200. Cost per person by car: 195
5 friends. Cost per person by plane: 200. Cost per person by car: 160
6 friends. Cost per person by plane: 200. Cost per person by car: 203
The optimal number of friends are: 5, and the total cost going by car is 800
```

Solution

```
#include <stdio.h>
#include <string.h>

#define USERS_PER_CAR 5
#define USERS_PER_ROOM 3

int main ()
{
    int costPlane, costCar, costHotel, costDinner, minFriends, maxFriends, totalCostCar, minCost, optFriends,
minTotalCost, persCostCar, i;
    scanf("%d %d %d %d %d %d", &costPlane, &costCar, &costHotel, &costDinner, &minFriends, &maxFriends);
    optFriends = minFriends;
    minCost = costPlane;
    minTotalCost = minCost * optFriends;
    for (i = minFriends; i <= maxFriends; i++)
    {
        totalCostCar = costCar * (int) ((i + USERS_PER_CAR - 1) / USERS_PER_CAR) + costHotel * (int)((i +
USERS_PER_ROOM - 1) / USERS_PER_ROOM) + costDinner * i;
        persCostCar = (double) totalCostCar / (double) i + .5;
        printf("%d friends. Cost per person by plane: %d. Cost per person by car: %d\n", i, costPlane, persCostCar);
        if (persCostCar < minCost)
        {
            optFriends = i;
            minCost = persCostCar;
            minTotalCost = totalCostCar;
        }
    }
    char transport[6];
    if (minCost < costPlane) strcpy(transport, "car");
    else strcpy(transport, "plane");
    printf("The optimal number of friends are: %d, and the total cost going by %s is %d\n", optFriends, transport,
minTotalCost);
    return 0;
}
```

21 The living pyramid

10 points

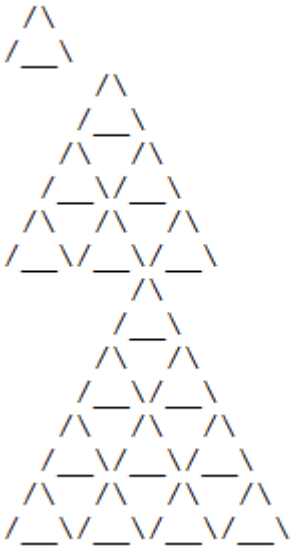
Introduction

The pharaoh Evaristo III was the greatest pyramid lover in the world.
In the very last day of his life, the pharaoh requested to build the most singular pyramid of the whole universe: the living pyramid. It is magical and unique because when someone shouts a positive integer number in front of it, the pyramid changes in order to have as many levels as the number. Thus your task is to create the most beautiful living pyramid.

Input

1 3 4

Output



Solution

```
#include <iostream>
#include <sstream>

int main ()
{

    std::string line;
    int n;

    getline(std::cin, line);
    std::istringstream iss(line);

    while (iss >> n)
    {
        int width = 4 * n; // Width of the grid
        int height = 2 * n; // Height of the grid
        int s = width/2 - 2; // Starting horizontal position
        for (int level = 1; level <= n; ++level, s -= 2) {
            // First row
            int cursor = 0;
            while (cursor < s) {
                std::cout << " ";
                ++cursor;
            }
            while (cursor < s + level*4) {
                std::cout << " /\\ ";
                cursor += 4;
            }
            std::cout << std::endl;

            // Second row
            cursor = 0;
            while (cursor < s) {
                std::cout << " ";
                ++cursor;
            }
            while (cursor < s + level*4) {
                std::cout << "/__\\";
                cursor += 4;
            }
            std::cout << std::endl;
        }
    }
}
```

22 Magic squares

10 points

Introduction

A magic square, is a n x n square grid filled with distinct positive integers in the range 1, 2, ..., n^2 such that each cell contains a different integer and the sum of the integers in each row, column and main diagonal is equal. This sum is called the magic constant of the magic square.
Even though magic square do not have a known application, and they belong to the recreational mathematics space, they have a long history, dating back to at least 650 BC in China. Many times, this squares have acquired magical or mythical significance, and have appeared as symbols in works of art.
In Europe, one of the most famous magic squares is the Albert Dürer order-4 magic square, immortalized in his 1514 engraving Melencolia I:

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

As you can see in the grid above, the magic constant 34 can be found in the rows, columns, and main diagonals. The two numbers in the middle of the bottom row give the date of the engraving: 1514. The numbers 1 and 4 at either side of the date correspond respectively to the letters “A” and “D”, which are the initials of the artist.

We have another famous magic square here, in Barcelona. The Passion façade of the Sagrada Familia features a 4x4 magic square.

The magic constant of the square is 33, the age of Jesus at the time of the Passion. 33 is also the number of the traditional degrees of Masonry, group with which Antonio Gaudi is usually related to.
As all we love mathematics, and we are now amazed about all this symbology represented by magic squares...

Write a program to check if a given square is magic or not.

Input

A series of rows with the corresponding columns of the magic square. As the order of the square might be variable the finish of the input will be marked with the character ‘#’.

Example 1
```
8 3 4
1 5 9
6 7 2
#
```

Example 2
```
8 13 2 11
1 12 7 14
15 6 16 4
10 3 9 5
#
```

Output

A string reporting whether the input is or is not a magic square.

Example 1
```
This is a magic square
```

Example 2
```
This is not a magic square
```

Solution

```
#include <iostream>
#include <sstream>
#include <vector>

using namespace std;

int addRow ( vector<vector<int> > magicSquare,  int row )
{
    int res = 0;

    for (int i = 0; i < magicSquare[row].size(); i++)
    {
        res += magicSquare[row][i];
    }

    return res;
}

int addCol ( vector<vector<int> > magicSquare, int col )
{
    int res = 0;

    for (int i = 0; i < magicSquare.size(); i++)
    {
        res += magicSquare[i][col];
    }

    return res;
}

int addDiag1 ( vector<vector<int> > magicSquare )
{
    int res = 0;

    for (int i = 0; i < magicSquare.size(); i++)
    {
        res += magicSquare[i][i];
    }

    return res;
}

int addDiag2 ( vector<vector<int> > magicSquare )
{
    int res = 0;

    for (int i = 0; i < magicSquare.size(); i++)
    {
        res += magicSquare[i][magicSquare.size()-1-i];
    }

    return res;
}

bool checkMagicSquare( vector<vector<int> > magicSquare )
{
    int rowRes = 0;
    int colRes = 0;
    int auxRes = 0;

    // Check rows and columns results
    for ( int i = 0; i < magicSquare.size(); i++ )
    {
        rowRes = addRow(magicSquare, i);
        colRes = addCol(magicSquare, i);

        if ( auxRes == 0 ) auxRes = rowRes;

        if ( ( rowRes != auxRes ) || ( colRes != auxRes ) )
        {
```

```
            return false;
        }
    }

    // Check main diagonals results
    if ( addDiag1(magicSquare) != auxRes )
    {
        return false;
    }
    if ( addDiag2(magicSquare) != auxRes )
    {
        return false;
    }

    // If we get this point we can ensure that the square
    // is a magic one
    //
    return true;
}

int main ()
{
    char aux[200];
    vector< vector<int> > magicSquare;
    vector<int> rowVector;

    // Get the magic square from the standard input
    // and fill the matrix
    //
    cin.getline(aux, 200);

    while (aux[0] != '#')
    {
        string s = aux;
        std::stringstream ss(s);
        int value;

        rowVector.clear();

        while(ss >> value)
        {
            rowVector.push_back(value);
        }

        magicSquare.push_back(rowVector);

        cin.getline(aux, 200);
    }

    // Do the real work
    bool isMagic = checkMagicSquare(magicSquare);

    if ( isMagic )
    {
        cout << "This is a magic square" << endl;
    }
    else
    {
        cout << "This is not a magic square" << endl;
    }

    return 0;
}
```

23 Roman calculator

10 points

Introduction

Maximilian lives in the Roman Empire and he has a grocery.
The business goes very well but he has a little problem: all the prices are in roman numerals and making sums with them is a painfully task.

Thank to Mercury, patron god of commerce, you are his friend, a well-known programmer in the empire.
Help to Maximilian making a roman calculator for attend customers faster.

Roman Numerals	
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

Input

The input will be a sequence of roman numerals ended with a +

```
I
II
IV
VI
+
```

Output

The output will be the sum of the given roman numerals, also in roman
The sum always will be less than 4000

```
XIII
```

Solution

```
#include <iostream>
#include <string>

unsigned int romanToDecimal(char roman)
{
    if ( roman == 'I' ) return 1;
    if ( roman == 'V' ) return 5;
    if ( roman == 'X' ) return 10;
    if ( roman == 'L' ) return 50;
    if ( roman == 'C' ) return 100;
    if ( roman == 'D' ) return 500;
    if ( roman == 'M' ) return 1000;
    return 0;
}

unsigned int romanToDecimal(std::string roman)
{
    unsigned int dec = 0;
    for ( int i = 0; i < roman.length(); i++ )
    {
        unsigned int digit = romanToDecimal(roman[i]);

        //If the next letter is bigger, the current one substarcts
        if ( i+1 < roman.length() && romanToDecimal(roman[i+1]) > digit )
        {
            dec = dec - digit;
        }
        //Otherwise it sums
        else
        {
            dec = dec + digit;
        }
    }
    return dec;
}

std::string decimalToRoman(unsigned int decimal)
{
    int thresholds[13] = {1,4,5,9,10,40,50,90,100,400,500,900,1000};
    std::string letters[13] = {"I","IV","V","IX","X","XL","L","XC","C","CD","D","CM","M"};

    std::string roman = "";
    while ( decimal !=  0)
    {
        for ( int i = 12; i >= 0; i-- )
        {
            if ( decimal >= thresholds[i] )
            {
                decimal = decimal - thresholds[i];
                roman = roman + letters[i];
                break;
            }
        }
    }
    return roman;
}

int main()
{
    unsigned int sum = 0;
    std::string line;
    while ( std::getline(std::cin, line) && line != "+" )
    {
        unsigned int decimal = romanToDecimal(line);
        sum = sum + decimal;
        //std::cout << line << ": " << decimal << ": " << decimalToRoman(decimal) << std::endl;
    }
    std::cout << decimalToRoman(sum) << std::endl;
    return 0;
}
```

24 We live in a spherical world

10 points

Introduction

Although lack of education, impossibility of access to intellectual works and some Christian expression of ideas affirming the Earth was flat... make it difficult to tell what wider population in early Middle Ages thought (if they even thought about this ;-), the fact is that the Earth is spherical. This idea first appeared in Greek philosophy with Pythagoras in the 6th century BC, and Aristotle provided the first observations supporting this idea, like the round Earth shadow over the Moon during lunar eclipses.

(*) Image not original from Aristotle

Flat trigonometry is not accurate when calculating distances between two points in the surface of the Earth, very important in navigation.
Luckily, the havesine formula allows us to calculate the shortest orthodromic distance or shorter distance between two points in the surface of a sphere.

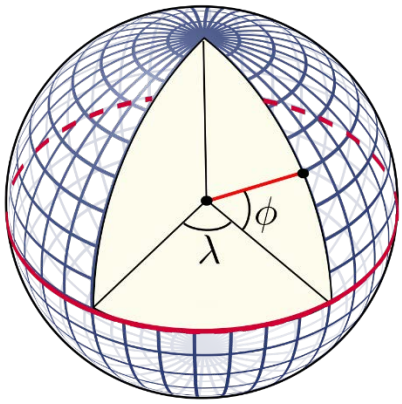
This can be very helpful, for example, if you are trying to win the Barcelona World Race, a non-stop, round-the-world yacht race, and look for the shortest route...
Looking at this map, it would seem that the horizontal distance on the Equator is similar to the one down on the 45ties latitude (in red, clipper route):

but a more realistic view, can help you see that maybe 'souther is shorter':

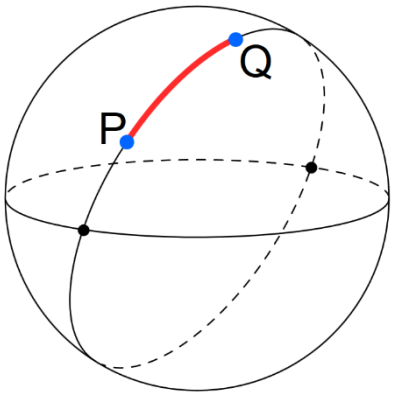
As an example, the circle of latitude 0°, that's the Equator, measures 40030Km (21600nm); while the Antarctic Circle, at latitude 66°33′ S, measures just 17662Km (9536nm)

Haversine formula

Latitude and longitude are basically nothing more than angles. Latitude is measured as your degrees north or south of the equator. Longitude is your degrees east or west of the prime meridian. The combination of these two angles pinpoints an exact location on the surface of the earth.



2- A perspective view of the Earth showing latitude (φ) and longitude (λ)



1- A diagram illustrating great-circle distance (drawn in red) between two points on a sphere, P and Q

As shown in the image *2* above, the quickest route between two points on the surface of the earth is a "great circle path" - in other words, a path that comprises a part of the longest circle you could draw around the globe that intersects the two points.

The shortest distance between two points on the globe can be calculated using the Haversine formula shown below.

$$d = 2r \arcsin\left(\sqrt{\operatorname{haversin}(\phi_2 - \phi_1) + \cos(\phi_1)\cos(\phi_2)\operatorname{haversin}(\lambda_2 - \lambda_1)}\right)$$
$$= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Where:

- *d* is the orthodromic distance between two points
- r is the radius of the sphere
- ϕ1, ϕ2: latitude of point 1 and latitude of point 2, in radians
- λ1, λ2: longitude of point 1 and longitude of point 2, in radians

Program specification

Your program should read two coordinates in latitude, longitude (float number in degrees), and calculate the orthodromic distance between them in meters, without decimals (truncated output).
We'll assume that Earth is perfectly spherical, with radius r=6371Km

Latitude will be positive for north hemisphere and negative for south.
Longitude will be positive for East and negative for West
(*) So, no indication for N/S or E/W is needed, and the calculation is direct from the values.

Input

```
latitude1,longitude1
latitude2,longitude2
```

Example

```
41.471485,2.094249
41.43166,2.126039
```

Output

```
distance
```

Example

```
5160
```

Solution

```
from math import radians, cos, sin, asin, sqrt

def haversine(point1, point2):
    """ Calculate the great-circle distance bewteen two points on the Earth surface.

    :input: two 2-tuples, containing the latitude and longitude of each point
    in decimal degrees.

    Example: haversine((45.7597, 4.8422), (48.8567, 2.3508))

    :output: Returns the distance bewteen the two points in kilometers
    """
    AVG_EARTH_RADIUS = 6371000  # in m

    # unpack latitude/longitude
    lat1, lng1 = point1
    lat2, lng2 = point2

    # convert all latitudes/longitudes from decimal degrees to radians
    lat1, lng1, lat2, lng2 = map(radians, (lat1, lng1, lat2, lng2))

    # calculate haversine
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = sin(lat * 0.5) ** 2 + cos(lat1) * cos(lat2) * sin(lng * 0.5) ** 2
    h = 2 * AVG_EARTH_RADIUS * asin(sqrt(d))
    return h  # in kilometers


# Read the input coordinates
# lat,lon
coor1=[float(x) for x in input().split(',')]
coor2=[float(x) for x in input().split(',')]

print(int(haversine(coor1,coor2)))
```

25 World chess championship

12 points

Introduction

Current world chess champion has requested a new chess-playing computer to train. The architecture of this chess-playing computer is composed by different modules and you are the developer of one important software part: the chess piece relative value module. This module will find out the value of the pieces that are still playing and the value of the ones already captured. Calculations of the value of pieces provide a rough idea of the state of play and help the computer to evaluate positions.

The chess pieces are assigned certain points to show how valuable they are. These points are shown in the table below:

The Queen	9 points
The Rook	5 points
The Bishop	3 points
The Knight	3 points
The Pawn	1 point

Obviously the value of the King is undefined as it cannot be captured during the course of the game.

The board is represented in an 8x8 two-dimensional array. Each array element would identify what piece occupied the given square, or altripletively, if the square is empty. A common encoding is to consider 0 as empty, positive as white, and negative as black, e.g., white pawn +1, black pawn −1, white knight +2, black knight −2, white bishop +3, and so on. Then the white pieces are identifed as follow:

1 white pawns
2 white knights
3 white bishops
4 white rooks
5 white queens
6 white king

Accordingly the black pieces are:

1 black pawns
2 black knights
3 black bishops
4 black rooks
5 black queens
6 black king

And this is the setup at the start of a game:

-4	-2	-3	-5	-6	-3	-2	-4
-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
4	2	3	5	6	3	2	4

Input

The input will be a sequence of 64 integer values representing the pieces in the board in a given moment of the game. For example:

```
0 0 0 0 0 -4 0 0 0 0 0 -6 0 0 0 0 0 -4 0 2 0 0 -1 -1 0 0 0 6 1 0 0 0 -1 0 0 0 0 0 1 1 0
-1 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Output

The output of the program report a couple of lines. The first one has two integer values and a list. The first represent the total relative value of the white pieces in game and the second the total relative value of the white pieces captured. The list will contain the white pieces captured following this order: pawns, knights, bishops, rooks and queen.
The second line has also two integer's values. The first represent the total relative value of the black pieces in game and the second the total relative value of the black pieces captured. The list will contain the black pieces captured following this order: pawns, knights, bishops, rooks and queen.
This is the output for the previous example:

```
11 28 [1, 1, 1, 1, 1, 2, 3, 3, 4, 5]
-14 -25 [-1, -1, -1, -1, -2, -2, -3, -3, -5]
```

Solution

```
import sys

# Main program (Python 3.5.2)

# 1. Parse data from input file
for line in sys.stdin:
    data = line.split()

# Number of each kind of pieces
numPawn = 8
numKnight = 2
numBishop = 2
numRook = 2
numQueen = 1
numKing = 1

# Id for each piece
idPawn = 1
idKnight = 2
idBishop = 3
idRook = 4
idQueen = 5
idKing = 6

# Value for each piece
valPawn = 1
valKnight = 3
valBishop = 3
valRook = 5
valQueen = 9

# Counters for pieces of each color
whitePawn = 0
whiteKnight = 0
whiteBishop = 0
whiteRook = 0
whiteQueen = 0
blackPawn = 0
blackKnight = 0
blackBishop = 0
blackRook = 0
blackQueen = 0

# Counters for total amount, white and black chess pieces
totalCnt = numPawn * valPawn + numKnight * valKnight + numBishop * valBishop + numRook * valRook + numQueen * valQueen
totalWhiteCnt = 0
totalBlackCnt = 0

# 2. Process the data getting the right value foreach piece
for i in data:
    # Convert from string to number
    num = int(i)

    # Identify the piece
    piece = abs(num)

    # Pawn
    if piece == idPawn:
        value = valPawn
        if num > 0:
            whitePawn += 1
        else:
            blackPawn += 1
    # Knight
    elif piece == idKnight:
        value = valKnight
        if num > 0:
            whiteKnight += 1
        else:
            blackKnight += 1
```

```
    # Bishop
    elif piece == idBishop:
        value = valBishop
        if num > 0:
            whiteBishop += 1
        else:
            blackBishop += 1
    # Rook
    elif piece == idRook:
        value = valRook
        if num > 0:
            whiteRook += 1
        else:
            blackRook += 1
    # Queen
    elif piece == idQueen:
        value = valQueen
        if num > 0:
            whiteQueen += 1
        else:
            blackQueen += 1
    # King
    else:
        value = 0

    if int(i)>0:
        totalWhiteCnt += value

    if int(i)<0:
        totalBlackCnt -= value

# 3. Printing results
# 3.1. For white pieces composing the list of the pieces captured

res = []
while whitePawn < numPawn:
    res.append(idPawn)
    whitePawn += 1
while whiteKnight < numKnight:
    res.append(idKnight)
    whiteKnight += 1
while whiteBishop < numBishop:
    res.append(idBishop)
    whiteBishop += 1
while whiteRook < numRook:
    res.append(idRook)
    whiteRook += 1
while whiteQueen < numQueen:
    res.append(idQueen)
    whiteQueen += 1

print (totalWhiteCnt, totalCnt - totalWhiteCnt, res)

# 3.2. For black pieces composing the list of the pieces captured

res = []
while blackPawn < numPawn:
    res.append(-idPawn)
    blackPawn += 1
while blackKnight < numKnight:
    res.append(-idKnight)
    blackKnight += 1
while blackBishop < numBishop:
    res.append(-idBishop)
    blackBishop += 1
while blackRook < numRook:
    res.append(-idRook)
    blackRook += 1
while blackQueen < numQueen:
    res.append(-idQueen)
    blackQueen += 1

print (totalBlackCnt, -(totalCnt + totalBlackCnt), res)
```

26 Goldbach's conjecture

14 points

Introduction

In 1742 the Prussian mathematician Christian Goldbach wrote a letter to his friend Leonhard Euler where he stated that:

"Any even number greater than 2 can be written as the sum of two prime numbers"

This statement is known as Goldbach's conjecture. Since then, mathematicians have been able to find such a pair of prime numbers for any even number greater than 2 that they've considered. It has been checked using computers for even numbers up to 4×10^{18}. Up to date nobody has been able to prove that this result holds for all even numbers. So, the conjecture remains unproven despite considerable effort.

Input

The input will be an even number greater than 4 and lesser than 5000. For example:

```
128
```

Output

The output of the program is a list of pair of prime numbers that sum up the number provided in the input. The list is ordered in increasing order considering the value of the first prime addend.
This is the output for the previous example:

```
19 + 109
31 + 97
61 + 67
```

Solution

```
#include <iostream>
#include <stdint.h>

bool isPrime( uint32_t x)
{
  bool prime = true;
  if (x == 1)
  {
    prime = false;
  }

  uint32_t i = 2;

  while (i*i <= x)
  {
    if (x % i == 0)
    {
      prime = false;
      break;
    }
    i += 1;
  }

  return prime;
}

int main()
{
  uint32_t num;

  std::cin >> num;

  if ((num <4) || (num % 2 != 0))
  {
      std::cout << 0 << std::endl;
  }
  else
  {
      for (uint32_t j=1; j<num; j++)
      {
          if (isPrime(j))
          {
              for (uint32_t k=j; k<num; k++)
              {
                  if (isPrime(k))
                  {
                      if (j+k == num)
                      {
                          std::cout << j << " + " << k << std::endl;
                      }
                  }
              }
          }
      }
  }
}
```

27 Time is money

14 points

Introduction

As you may already know, HP has an R&D site located in Leon, Spain.
It is usual that people there travel to our site in Sant Cugat in order to catch up with team mates, attend to learning sessions, events, etc.
The cheapest way to get from Leon to Barcelona is by taking a train.
Unfortunately, railways in Spain are notorious for their delays, and this is annoying for passengers. The train company has signed a punctuality commitment and offers refunds for clients. You are told to develop a software program that calculates the amount to be repaid to each passenger, which depends on how much time the train was delayed, and the price that they paid. The code also has to compute the total money loss for the company.

The route of the train is as follows:
Leon → Palencia - Burgos - MirandaDeEbro - Vitoria - Pamplona - Tudela - Zaragoza -Lleida - Tarragona → Barcelona

The refund policy is as follows:
- A compensation of half the price of the ticket will be given if the train is delayed 30 to 59 minutes.
- A compensation of the entire price of the ticket will be given if the train is delayed 60 or more minutes.

Note that a passenger only has right to refund if the delay occurs in the part of the route that he travels, or before. Someone that goes from Burgos to Vitoria will not receive any money if the train suffers a delay between Zaragoza and Lleida.
However, if there is a delay between Leon and Palencia, that one affects him, since the train will arrive late at the station of departure (Burgos).

Input

The input is a report from the company, and consists of 3 parts:
-The amount of delays and passengers that complained
- A list of the delays, specifying where occurred and how much time (minutes) was lost*.
-A list of the complaining passengers, including the name, city of departure, city of destination, and the price (in euros) of the ticket.

*To simplify the problem, it is assumed that there are only delays between stations. Then, the city appearing in the delay information indicates that the incident was between that one and the previous one, e.g: "Burgos 30" means that there was a delay of 30 minutes between Palencia and Burgos.

Example 1

```
2 4
MirandaDeEbro 40
Zaragoza 35
Juan Leon Burgos 60
Maria Pamplona Zaragoza 100
Luis Palencia Barcelona 140
Ana Tarragona Barcelona 20
```

Example 2

```
2 1
MirandaDeEbro 40
Zaragoza 35
Michael Palencia Vitoria 18
```

Output

The output must be a list of sentences indicating how much money Renfe has to pay each passenger, and a final line telling the total amount of money that was spent in compensations. Use exactly the same text format than indicated in the example below.
The output for the previous input would be:

Example 1

```
Juan receives 0 euros
Maria receives 100 euros
Luis receives 140 euros
Ana receives 20 euros
Total loss for train company is: 260 euros
```

Example 2

```
Michael receives 9 euros
Total loss for train company is: 9 euros
```

Solution

```
# DATA DEFINITIONS:

stops = ['Leon','Palencia','Burgos','MirandaDeEbro','Vitoria','Pamplona','Tudela','Zaragoza','Lleida',
         'Tarragona','Barcelona']

delays = {}
passengers = []
origins = []
destinations = []
prices = []
totalLoss = 0

# GET THE INFORMATION FROM INPUT:
firstLine = raw_input()
info = firstLine.split()
nDelays = int(info[0])
nPassengers = int(info[1])

# Delays info:
for i in range (0,nDelays):
    line = raw_input()
    info = line.split()
    delays[info[0]] = int(info[1])

# Delays info:
for i in range (0, nPassengers):
    line = raw_input()
    info = line.split()
    passengers.append(info[0])
    origins.append(info[1])
    destinations.append(info[2])
    prices.append(int(info[3]))

# AUXILIARY FUNCTIONS:
def check_delay(destination):

        indexDest = stops.index(destination)           # Find the index of the destination city

        delayTime = 0

        for delayPlace in delays.keys():               # Iterate over delays
                indexDelay = stops.index(delayPlace)       # Find the index of the delay place
                if (indexDelay <= indexDest):              # If the delay place is before the destination, it affects
passenger.
                        delayTime += delays[delayPlace];       # Count the delay time

        return delayTime

# MAIN: CALCULATE REFUNDS:
for i in range (0,len(passengers)):                # Iterate over passenger list

        delayTime = check_delay(destinations[i])       # Calculate the dtotal elay of the train

        refundFactor = 0.0

        if delayTime > 30:                             # Decide how much money to give back
                refundFactor = 0.5
                if delayTime > 60:
                        refundFactor = 1

        refund = refundFactor*prices[i]                # The refund depends on the ticket price

        print ("%s receives %d euros" %(passengers[i], refund))

        totalLoss += refund                            # Add this refund to the summatory

print ("Total loss for train company is: %d euros" %totalLoss)
```

28 Shuttle bus service schedule

16 points

Introduction

You just started a new job in the customer services department for a hotel in the exotic island of Koh Tao and your first assignment is to plan the shuttle bus service between the airport and the hotel.

Each day you will be given a list of guest families with their arrival time and the number of family members. The hotel management will indicate how many bus trips you can arrange for the day. Fortunately, buses are so big that each of them has enough capacity to carry all the guests in one go if needed.

Your job is to schedule the bus departures to minimize the hosts' waiting time and report the best possible total time.

Input format:
[Number of bus trips, not more than 10]
[Number of guest families, not more than 10]
[Arrival time] [Number of family members]

Output format:
Total waiting time: [waiting time in minutes] minutes

Input

```
2
3
5:00 5
10:00 2
20:00 1
```

Output

```
Total waiting time: 1200 minutes
```

In this example, we have 3 families arriving to the island at 5:00, 10:00 and 20:00 but we are only allowed to schedule 2 bus departures.

The best solution consists in scheduling the first departure at 5:00 and the second one at 20:00. By doing so, the 1st and the 3rd family won't have to wait at all, and the 2 members of the 2nd family will have to wait 10 hours each for a total of 20h waiting hours (10h*2p=1200 minutes).

Any other schedule using 2 buses would increase the total waiting time per person. For example, if we had scheduled the buses at 10:00 and 20:00, the total waiting time would have been 25h (5h*5p=1500 minutes), which is worse than the 20 hours obtained with the best possible schedule.

Solution

```
import java.util.Scanner;

public class Main {

	private static Arrival[] arrivals = null;
	private static int minTime = Integer.MAX_VALUE;

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);

		final int BUSES = sc.nextInt();
		final int ARRIVALS = sc.nextInt(); sc.nextLine();
		arrivals = new Arrival[ARRIVALS];

		// Parse input
		for (int i=0;i<ARRIVALS;i++)
		{
			String line = sc.nextLine();
			String[] tokens = line.split(" |:");
			arrivals[i] = new Arrival(Integer.parseInt(tokens[0])*60+Integer.parseInt(tokens[1]),
Integer.parseInt(tokens[2]));
		}

		// "choices" indicates our current bus schedule decisions bitmap:
		//     'true' for the arrivals in which we will schedule a bus.
		//     'false' otherwise.
		boolean[] choices = new boolean[ARRIVALS];
		choices[ARRIVALS-1] = true; // We must schedule a bus for the last family since we don't want to let
them waiting forever.
		recurse(ARRIVALS-2, BUSES-1, choices); // Start recursion backwards from the latest arrival to the
first one.

		System.out.println("Total waiting time: "+minTime+" minutes");
		sc.close();
	}



	private static void recurse(int arrivalIndex, int busesLeft, boolean[] choices)
	{
		//System.out.println("index: "+arrivalIndex+" - busesLeft: "+busesLeft); printChoices(choices);

		if (busesLeft==0 || arrivalIndex<0)  // We have no more buses in this branch or we have already
considered all arrivals. Stop recursion.
		{
			int time = computeWaitingTime(choices);
			if (time<minTime)
			{
				minTime = time;
				//printChoices(choices);
			}
		}
		else  // Recursion case: should we place a bus for this arrival? (true/false)
		{
			// We choose to place a bus at this arrival time.
			choices[arrivalIndex]=true;
			recurse(arrivalIndex-1, busesLeft-1, choices);
			choices[arrivalIndex]=false;  // Undo changes.

			// We have less buses that arrivals pending to consider, so we need to consider not placing a
bus for this arrival
			if (arrivalIndex+1>busesLeft)
			{
				// We choose not to place a bus at this arrival time (choices[arrivalIndex]=false
already)
				recurse(arrivalIndex-1, busesLeft, choices);
			}
		}
	}
```

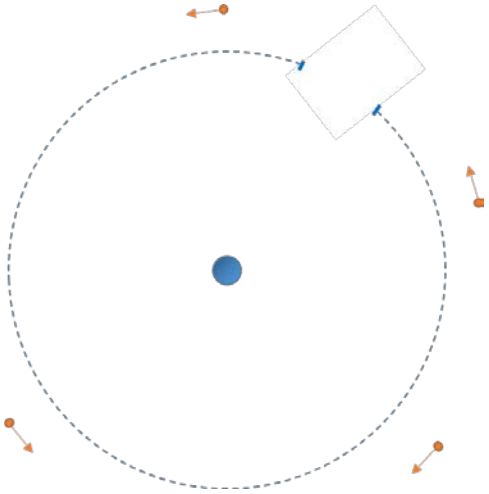
```
        private static int computeWaitingTime(boolean[] choices)
        {
                int cost=0;
                // Compute the waiting time for all families.
                for (int i=0;i<arrivals.length-1;i++)
                {
                        int familyWait = 0;
                        for (int j=i;j<choices.length && !choices[j];j++)
                                familyWait += arrivals[j+1].minutes-arrivals[j].minutes;
                        familyWait *= arrivals[i].guests;
                        cost+=familyWait;
                }
                return cost;
        }

        private static void printChoices(boolean[] choices)
        {
                System.out.print("Choices:");
                for (int i=0;i<choices.length;i++)
                        System.out.print(" "+(choices[i]?"1":"0"));
                System.out.println(" - minTime="+minTime);
        }

        // Contains arrival information as a pair.
        private static class Arrival
        {
                public int minutes;  // Day time in minutes (e.g. 1:02am = 62 minutes).
                public int guests;   // Number of guests
                public Arrival(int minutes,int guests){
                        this.minutes = minutes;
                        this.guests = guests;
                }
        }
}
```

29 Transmission Tower

16 points

Introduction

Hi commander! You are our last hope. You are in the planet Scarif, in the middle of a battle and your mission is to send the blueprints of the Death Star to the rebels.
You already have the blueprints and you are in the transmission tower, but we have a problem. The planet is surrounded by a shield that does not permit any communication with the ships that orbit the planet.
One of our ships have opened a hole in the shield, but we don't know how long we can have it open.



You must choose a ship to transmit the message in the shortest possible time. You have the following data:

- Size of the blueprints in terabytes (TB)
- Coordinates of the hole (start and end) in degrees
- Number of ships orbiting the planet
- Name of each ship orbiting the planet
- Angular position of each ship orbiting the planet in degrees.
- Angular speed of each ship orbiting the planet in rad/sec
- Bandwidth of each ship orbiting the planet in MB/sec

You also have a robot that provides you the next mathematical information:

1. To translate from degrees to radians you can use the formula:

$$rad = \frac{degrees * \pi}{180}$$

2. To calculate the time to go from an angular coordinate X to another coordinate Y you can use the formula:

$$t = \frac{Y - X}{w}$$

Where:

- X is the initial position in radians
- Y is the final position in radians
- w is the angular speed of the ship in radians/sec
- t is the time to arrive to Y in seconds

The conversion between storing units is:

1 TB = 1024 GB
1 GB = 1024 MB

The transmission must be continuous, if the ship goes out of the whole the transmission must be started again from the start when the ship enters again in the whole.

If you find a ship to transmit the blueprints your program must output the name of this ship. If there are not any ship with enough bandwidth to transmit the message your program must output the message **MISSION FAILED**.

Input

```
150
67;100
8
Droid starfighter;45;0.0004;6787
ARC-170;123;0.0055;27657
Delta-7;98;0.0067;477889
TIE interceptor;176;0.000999;83776
Naboo N-1;356;0.000134;8976
Y-wing;8;0.000033;23907
A-wing;222;0.0055;43456
X-wing;90;0.00397;98000
```

Output

```
Y-wing
```

Solution

```
import math

def DegreeToRad(degrees):
    return degrees * math.pi / 180

class Ship:
    def __init__(self, name, positionInDegrees, angularSpeed, bandwidth):
        self.Name = name
        self.Position = DegreeToRad(positionInDegrees)
        self.AngularSpeed = angularSpeed
        self.BandWidth = bandwidth

class ShipExtraInfo:
    def __init__(self, ship, wholeStart, wholeEnds, dataSizeInMB):
        self.Ship = ship
        self.TransmissionTime = dataSizeInMB / ship.BandWidth
        self.TimeInWhole = self.CalculateTimeInTheWhole(ship, wholeStart, wholeEnds)
        self.TimeToEnterInTheWhole = self.CalculateTimeToEnterInTheWhole(ship, wholeStart, wholeEnds)
        self.TimeToFinishTransmission = self.TransmissionTime + self.TimeToEnterInTheWhole

    def GetTransmissionTime(self):
        return self.TransmissionTime

    def GetTimeInWhole(self):
        return self.TimeInWhole

    def GetTimeToEnterWhole(self):
        return self.TimeToEnterInTheWhole

    def GetTimeToFinishTransmission(self):
        return self.TimeToFinishTransmission

    def CalculateTimeToEnterInTheWhole(self, ship, start, finish):
        time = 0
        if self.IsCoordinateBetweenStartAndFinish(start, finish, ship.Position):
            time = 0
        else:
            if ship.AngularSpeed > 0:
                time = self.CalculateTimeToPosition(ship.Position, ship.AngularSpeed, finish)
            else:
                time = self.CalculateTimeToPosition(ship.Position, ship.AngularSpeed, start)

        return time


    def CalculateTimeInTheWhole(self, ship, start, finish):
        timeInsideTheWhole = 0
        timeToWholeStart = self.CalculateTimeToPosition(ship.Position, ship.AngularSpeed, start)
        timeToWholeFinish = self.CalculateTimeToPosition(ship.Position, ship.AngularSpeed, finish)

        if self.IsCoordinateBetweenStartAndFinish(start, finish, ship.Position):
            if ship.AngularSpeed > 0:
                timeInsideTheWhole = timeToWholeFinish
            else:
                timeInsideTheWhole = timeToWholeStart
        else:
            if ship.AngularSpeed <= 0:
                timeInsideTheWhole = timeToWholeStart - timeToWholeFinish
            else:
                timeInsideTheWhole = timeToWholeFinish - timeToWholeStart

        return timeInsideTheWhole

    def IsCoordinateBetweenStartAndFinish(self, start, finish, coordinate):
        return coordinate >= start and coordinate <= finish

    def CalculateTimeToPosition(self, startPosition, speed, finishPosition):
        if finishPosition < startPosition:
            finishPosition += 2*math.pi
        return (finishPosition - startPosition) / speed
```

```
    def CanDataBeTotallyTransmitted(self):
        return self.TransmissionTime <= self.TimeInWhole

    def ToString(self):
        return self.Ship.Name + "; " + str(self.Ship.Position) + "; " + str(self.Ship.AngularSpeed) + "; " +
str(self.Ship.BandWidth)


def main():

    #Parse inputs

    dataInMB = int(float(raw_input())) * 1024 * 1024

    strWhole = raw_input()
    wholeCoordinatesInStr = strWhole.split(";")

    wholeStart = DegreeToRad(int(float(wholeCoordinatesInStr[0])))
    wholeFinish = DegreeToRad(int(float(wholeCoordinatesInStr[1])))

    numberOfShips = int(float(raw_input()))

    shipsList = []
    for i in range(numberOfShips):
        shipData = raw_input().split(";")
        ship = Ship(shipData[0], int(float(shipData[1])), float(shipData[2]), int(float(shipData[3])))
        shipsList.append(ShipExtraInfo(ship, wholeStart, wholeFinish, dataInMB))

    validShips = []
    for ship in shipsList:
        if ship.CanDataBeTotallyTransmitted():
            validShips.append(ship)

    # Sort the list
    validShips = sorted(validShips, key=lambda ship: ship.GetTimeToFinishTransmission())

    if len(validShips) > 0:
        print validShips[0].Ship.Name
    else:
        print "MISSION FAILED"
```

30 Hero of the Core

18 points

Introduction

You work in the upcoming amazing game called Hero of the Core, which intends to reinvent the Multiplayer online battle arena (MOBA) genre. The game will put you in a 2D arena map full of towers and enemy heroes. As one of the last members of your clan you should defend your core at all cost besides your comrades, destroying the enemy heroes and their towers.

The main designer has requested to develop a small prototype to show the base game mechanics to some investors. In this early prototype the objective will be to test the movement and destroy mechanics against the towers and the enemy heroes, but there will be only one player hero and your core will be safe as the enemy heroes won't move.

Each time you destroy a tower you will win a weapon to destroy an enemy hero, if you try to defeat an enemy hero without a weapon you will be defeated! So the number of enemy heroes and towers need to be the same.

The game will end with victory when all the towers and enemies are defeated, but it will end with a defeat if your hero perishes against an enemy hero, in any case, The result will always be shown after show the status.

If there is an error creating the map, the game will show the status, the error reason and will exit immediately:

- If there are no hero or core the next message should be shown:
  - **Error: It has to be a hero and a core!**
- If more than 1 hero the next message should be shown:
  - **Error: It can't be several main heroes!**
- If more than 1 core the next message should be shown:
  - **Error: It can't be several cores!**

Example 1 - Input

The input to your game will be:

- The size of the arena given as maximum rows and columns.
- Several positions in the map, each of them with only one of the following elements:
  - Empty - represented as '_', by default all arena is empty.
  - Hero player - represented as 'h', in this prototype it has to be only one.
  - Core - represented as 'c', it has to be only one!
  - Enemy Tower - represented as 't'
  - Enemy hero – represented as 'e'
- A dot, this will help to delimitate arena setup from movements.
- A sequence of movements given as a pair (±1, ±1), indicating the rows and columns to advance each round.
  - Your hero can only walk one position at a time in any direction and without exit the map.
    - Given a (2,2) movement the output should show the next message and keep reading the next input:
      - ***Can't move that distance! (2,2)***
  - If your hero walks into an enemy tower position, the tower will be destroyed and the hero collects a weapon.
  - If your hero walks into an enemy position,
    - If the hero has a weapon he will beat the enemy and replace its position.
    - If the hero has no weapon the game ends and should show the string:
      - ***You died!***
  - If there are no more movements and the game has not ended in any way (win, hero dies, etc) the next message should be shown:
    - ***No more movements! Game ended without result!***

```
3 3          ← this is a 3x3 arena
0 1 h        ← row 0 and column 1 is the hero position
0 0 c        ← row 0 and column 0 is the core position
1 1 e        ← at row 1 and column 1 there is an enemy hero
1 2 t        ← at row 1 and column 2 there is a tower
.            ← From now on all inputs are movements!
1 1          ← Go down one row and right one column
0 -1         ← Go left one column
```

Example 1 - Output

The output will be a matrix, showing each element and the evolution of the movements.

```
Game Status:
 c  h  _

 _  e  t

 _  _  _

[Enemy heroes left=1][Current weapons=0]
 c  _  _

 _  e  h

 _  _  _

[Enemy heroes left=1][Current weapons=1]
 c  _  _

 _  h  _

 _  _  _

[Enemy heroes left=0][Current weapons=0]
You have cleaned the battle arena! Congratulations!
```

Example 2 - Input

```
3 3
0 1 h
0 0 c
1 1 e
1 2 t
.
1
```

Example 2 - Output

```
Game Status:
 c  h  _

 _  e  t

 _  _  _

[Enemy heroes left=1][Current weapons=0]
 c  _  _

 _  e  h

 _  _  _

[Enemy heroes left=1][Current weapons=1]
No more movements! Game ended without result!
```

Solution

```
import java.util.*;

public class Main {

		static final char EMPTY = '_';
		static final char HERO = 'h';
		static final char CORE = 'c';
		static final char TOWER = 't';
		static final char ENEMY = 'e';
		static char[][] map;
		static int enemiesDefeated,currentWeapons;
		static int totalEnemies,totalWeapons;
		static boolean dead;
		static int heroRow= -1, heroCol= -1;
		static int coreRow= -1, coreCol= -1;

	public static void main(String[] args) {

		Scanner sc = new Scanner(System.in);
		StringTokenizer st;
		String line="";
		totalEnemies = enemiesDefeated = 0;
		totalWeapons = currentWeapons = 0;
		int maxRows,maxCols=0;

		//Create map
		st = new StringTokenizer(sc.nextLine());
		maxRows = Integer.parseInt(st.nextToken());
		maxCols = Integer.parseInt(st.nextToken());
		map = new char[maxRows][maxCols];
		//populate empty map
		for (int i=0; i < maxRows ; i++ ) {
			for (int j=0; j < maxCols ; j++ ) {
				map[i][j]=EMPTY;
			}
		}

		String errorMessage = null;
		//Populate map with elements
		line = sc.nextLine();
		while(!line.equals("."))
		{
			st = new StringTokenizer(line);
			int row = Integer.parseInt(st.nextToken());
			int col = Integer.parseInt(st.nextToken());
			map[row][col]=st.nextToken().charAt(0);
			if(checkPosition(ENEMY,row,col))
				totalEnemies++;
			else if(checkPosition(TOWER,row,col))
				totalWeapons++;
			else if(checkPosition(HERO,row,col)) {
				if(heroRow !=-1) {
					errorMessage = "Error: It can't be several main heroes!";
				} else {
					heroRow=row;
					heroCol=col;
				}
			} else if(checkPosition(CORE,row,col)) {
				if(coreRow !=-1) {
					errorMessage = "Error: It can't be several cores!";
				} else {
					coreRow=row;
					coreCol=col;
				}
			}
			line = sc.nextLine();
		}

		if(heroRow==-1 || coreRow==-1) {
			errorMessage = "Error: It has to be a hero and a core!";
```

```
		}

		if(totalEnemies!=totalWeapons) {
			errorMessage = "Error: It has to be the same number of enemies and towers!";
		}

		System.out.println("Game Status:");
		//Read and execute movements
		while(true)
		{
			printMap();
			printStatus();
			String nextLine = null;
			try {
			 nextLine = sc.nextLine();
			} catch (java.util.NoSuchElementException e) {
					nextLine = null;
			}
			if(isFinished(nextLine,errorMessage))
				break;

			st = new StringTokenizer(nextLine);
			int despRow = Integer.parseInt(st.nextToken());
			int despCol = Integer.parseInt(st.nextToken());
			execMove(despRow,despCol);
		}
	}

	static boolean isFinished(String line, String errorMessage) {
		if(enemiesDefeated==totalEnemies){
			System.out.println("You have cleaned the battle arena! Congratulations!");
			return true;
		}else if(dead) {
			System.out.println("You died!");
			return true;
		}else if(line == null || line =="") {
			System.out.println("No more movements! Game ended without result!");
			return true;
		} else if (errorMessage != null) {
			System.out.println(errorMessage);
			return true;
		}
		return false;
	}
	static void execMove(int despRow, int despCol)
	{
		if(despRow > 1 || despCol > 1 || despRow < -1 || despCol < -1) {
			System.out.println("Can't move that distance! ("+despRow+","+despCol+")");
		} else {
			int newRow = heroRow+despRow;
			int newCol = heroCol+despCol;
			if(checkPosition(TOWER,newRow,newCol)) {
				currentWeapons++;
			} else if(checkPosition(ENEMY,newRow,newCol)) {
				if(currentWeapons==0)
					dead = true;
				else {
					currentWeapons--;
					enemiesDefeated++;
				}
			}
			map[heroRow][heroCol] = EMPTY;
			map[newRow][newCol] = HERO;
			heroRow=newRow;
			heroCol=newCol;
			//System.out.println("Last movement ("+despRow+","+despCol+")");
		}
	}

	static boolean checkPosition(char value,int row, int col) {
		return map[row][col]==value? true:false;
	}
```

```
        static void printStatus() {
                System.out.println("[Enemy heroes left="+(totalEnemies-enemiesDefeated)+"][Current
weapons="+currentWeapons+"]");
        }

        static void printMap() {
                for (int i=0; i < map.length ; i++ ) {
                        for (int j=0; j < map[0].length ; j++ ) {
                                System.out.print(map[i][j]);
                                if(j != map[0].length-1)
                                        System.out.print(" ");
                        }
                        System.out.println();
                }
        }

}
```

31 The CodeWars is MINE!

20 points

Introduction

Lets say you are in programming competition in which you have some hours to complete a certain amount of problems to solve. Each problem have some points you will earn in case of being solved, and solving more complicated problems will earn you more points.
During the competition welcome, you and your team realize that there are many good programmers, and you have some doubts whether you could win. Therefore, making use of your most notorious ability, strategy, you have decided to write a program that will tell you which problems you should solve to maximize your chances to win.
You will be given the number of hours you have to complete the problems, the points you might earn and the time estimated to complete each problem. So your task is to decide which problems you will solve, the time it will require, and the total points you will earn, taking into consideration some key points.

- Based on the experience of previous CodeWars editions, you know that:
  - In average, the CodeWars servers are down during 5 minutes every hour of competition. During this down time you will not be able to continue solving problems, that is, this minutes are wasted.
  - In average, 2 question tickets are opened every 5 problems, and it takes 5 minutes answering to each of these tickets. During the resolution of the tickets, you will not be able to continue solving problems, and thus, this time is wasted too.
- To make sure you do not face big difficulty changes and that you are able to solve the problems, you decide to only solve consecutive problems.
- In case more than one combination of problems give the highest score, you will prioritize the combination that takes less time, and if they are still equal, the combination with the easiest problems to solve.

Input & Output:
The first input line will contain the number of hours the competition lasts and the second line the total number of problems available in the competition. Afterwards, one line per problem containing the problem number, the points earned if the problem is solved and the estimated time it will require to solve it in minutes. All these parameters are given as a natural numbers.
Input format:
H
N

1 P(1) T(1)
2 P(2) T(2)
...
N P(N) T(N)
Output format:
Maximum points to achieve: SCORE, by solving P problem(s):
p(x) p(x+1) p(x+2) p(x+3) p(x+P)

Input

```
1
8
1 1 5
2 1 5
3 2 10
4 3 15
5 5 18
6 8 20
7 13 25
8 21 35
```

Output

```
Maximum points to achieve: 21, by solving 1 problem(s):
8
```

Solution

```
#include <iostream>
#include <vector>
using namespace std;

typedef unsigned int uint;

struct Problem
{
    uint points;
    uint time;
};

struct HistoBin
{
    uint pts;
    uint time;
    uint numPr;
};


////////////////////////////////////////////////////////////////////////////////
//
// Aux methods headers
//
////////////////////////////////////////////////////////////////////////////////

void skipProblemsServerDown( const vector<Problem> & problems, vector<HistoBin> & histo, uint bin, uint maxTime, uint
currTime );

void skipProblemsTicketing( const vector<Problem> & problems, vector<HistoBin> & histo, uint bin, uint maxTime, uint
currTime );

#ifdef DEBUG
void printHisto( const vector<HistoBin> & histo );
```

```
#endif


////////////////////////////////////////////////////////////////////////////////
//
// Main
//
////////////////////////////////////////////////////////////////////////////////

int main( int argc, char *argv[] )
{
    uint H, N, id;
    cin >> H >> N;

    // Change CodeWars duration from hours to minutes
    //
    H *= 60;

    // Storage for all problems
    //
    #ifdef DEBUG
    cout << "Time = " << H << ", #Pr = " << N << endl;
    #endif
    vector<Problem> problems = vector<Problem>( N );

    // Reading problems info
    //
    for ( uint i = 0; i < N; i++ )
    {
        cin >> id >> problems[i].points >> problems[i].time;
    }

    // As problems must be consecutive, build an histogram of the total points
    // that can be achieved starting from each problem
    //
    vector<HistoBin> histo( N );
    for ( int i = N - 1; i >= 0; i-- )
    {
        uint totalPts  = 0;
        uint totalTime = 0;
        uint totalPr   = 0;
        for ( int j = i; j >= 0; j-- )
        {
            // Check if it is possible to add this problem
            //
            if ( ( H - totalTime ) > problems[j].time )
            {
                totalPts  += problems[j].points;
                totalTime += problems[j].time;
                totalPr++;
            }
            else
            {
                // It is not possible to complete this problem
                // just break the loop.
                //
                break;
            }
        }

        // Update histogram with data calculated starting from problem 'i'
        //
        histo[i].pts   = totalPts;
        histo[i].time  = totalTime;
        histo[i].numPr = totalPr;
    }

    #ifdef DEBUG
    cout << endl << "Histogram" << endl;
    printHisto( histo );
    #endif
```

```
    ////////////////////////////////////
    // Server down calculation
    //

    // For each calculation, it is necessary to take into account the time wasted
    // for server down time and ticket resolution
    //
    uint server   = 0;
    uint ticket   = 0;
    //uint overtime = 0;
    for ( int i = N - 1; i >= 0; i-- )
    {
        // First take server down time into cosideration. This migh force to skip
        // problems
        //
        server   = 5 * ( H / 60 );
        histo[i].time += server;
    }
    #ifdef DEBUG
    cout << endl << "Histogram - SERVER DOWN EXCEEDED" << endl;
    printHisto( histo );
    #endif
    for ( int i = N - 1; i >= 0; i-- )
    {
        if ( histo[i].time > H )
        {
            // It is necessary to skip problems
            //
            skipProblemsServerDown( problems, histo, i, H, histo[i].time );
        }

    }
    #ifdef DEBUG
    cout << endl << "Histogram - SERVER DOWN CORRECTED" << endl;
    printHisto( histo );
    #endif

    //
    ////////////////////////////////////


    ////////////////////////////////////
    // Ticket resolution
    //

    for ( int i = N - 1; i >= 0; i-- )
    {
        // Now that the number of problems is updated, we need to add ticketing time
        //
        //                                      ( 2 ticket / 5 pr ) * 5 min
        //ticket   = uint( double(histo[i].numPr) * ( 2.0 / 5.0 ) * 5.0 )
        //         = double(histo[i].numPr) * 2.0
        //         = histo[i].numPr << 1
        ticket         = histo[i].numPr << 1;
        histo[i].time += ticket;
    }
    #ifdef DEBUG
    cout << endl << "Histogram - TICKETING EXCEEDED" << endl;
    printHisto( histo );
    #endif
    for ( int i = N - 1; i >= 0; i-- )
    {
        if ( histo[i].time > H )
        {
            skipProblemsTicketing( problems, histo, i, H, histo[i].time );
        }

    }
    #ifdef DEBUG
    cout << endl << "Histogram - TICKETING CORRECTED" << endl;
    printHisto( histo );
    #endif
```

```
    //
    ////////////////////////////////////


    // Once re-calculation is finised, find which combination gives highest score
    //
    uint maxPtsId = 0;
    uint maxPtsTm = H + 1;
    for ( uint i = 1; i < N; i++ )
    {
        if ( histo[i].pts > histo[maxPtsId].pts )
        {
            maxPtsId = i;
            maxPtsTm = histo[i].time;
        }
        else if ( histo[i].pts == histo[maxPtsId].pts )
        {
            // In case they are equal in points, priorize which takes less time
            if ( histo[i].time < histo[maxPtsId].time )
            {
                maxPtsId = i;
                maxPtsTm = histo[i].time;
            }

            // In case they are still equal, priorize that with easiest problems
            // but as we  accessing vector in ascending order, 'i' can never be
            // smaller than 'maxPtsId'
        }
    }


    // Now that we know the winner combination, just output in proper format
    //
    cout << "Maximum points to achieve: " << histo[maxPtsId].pts << ", by solving " << histo[maxPtsId].numPr << "
problem(s):" << endl;
    uint i = ( maxPtsId - histo[maxPtsId].numPr + 1 );
    cout << i + 1;
    i++;
    for ( ; i <= maxPtsId; i++ )
    {
        cout << " " << i + 1;
    }
    cout << endl;


    return 0;
}


////////////////////////////////////////////////////////////////////////////////
//
// Aux methods
//
////////////////////////////////////////////////////////////////////////////////

void skipProblemsServerDown( const vector<Problem> & problems, vector<HistoBin> & histo, uint bin, uint maxTime, uint
currTime )
{
    // Sanity
    if ( currTime <= maxTime ) { return; }

    int exceedingTime = currTime - maxTime;

    // Navigate throught this problem secuence in ascending order, as the first ones
    // require less time to complete and give less points.
    for ( uint i = ( bin - histo[bin].numPr + 1 ); i <= bin && exceedingTime > 0; i++ )
    {
        exceedingTime   -= problems[i].time;

        histo[bin].time -= problems[i].time;
        histo[bin].pts  -= problems[i].points;
        histo[bin].numPr--;
    }
```

```
}

void skipProblemsTicketing( const vector<Problem> & problems, vector<HistoBin> & histo, uint bin, uint maxTime, uint
currTime )
{
    int exceedingTime = currTime - maxTime;
    uint numPrSkipped = 0;

    // Sanity
    if ( currTime < maxTime ) { return; }

    // Based on the exceeding time, it is necessary to calculate how many
    // problems we need to skip
    //
    // It takes 2 tickets of 5 min per 5 problems ~ 10 min per 5 problems ~ 2 min per problem ~ (pr * 2) min

    for ( uint i = ( bin - histo[bin].numPr + 1 ); i <= bin && exceedingTime > 0; i++ )
    {
        exceedingTime   -= problems[i].time;

        histo[bin].time -= problems[i].time;
        histo[bin].pts  -= problems[i].points;
        histo[bin].numPr--;

        numPrSkipped++;
    }

    // For each skipped problem, we no not have to take into account the estimated
    // time wasted for ticketing. 2 min per skipped problem.
    //
    histo[bin].time -= ( numPrSkipped << 1 );
}

#ifdef DEBUG
void printHisto( const vector<HistoBin> & histo )
{
    cout << "Pr\tPts\t#Pr\tTime" << endl;
    for ( uint i = 0; i < histo.size(); i++ )
    {
        cout << i << "\t" << histo[i].pts << "\t" << histo[i].numPr << "\t" << histo[i].time << endl;
    }
    cout << endl;
}
#endif
```

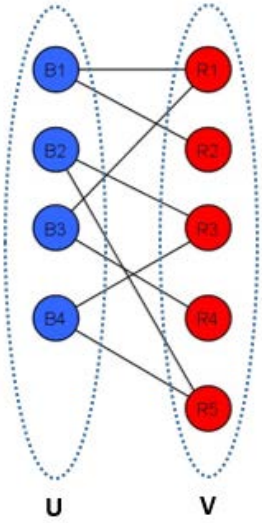
32 Two-colorable graphs

22 points

Introduction

In the mathematical field of graph theory, a bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint sets U and V (that is, U and V are each independent sets) such that every edge connects a vertex in U to one in V.

The two sets U and V may be thought of as a coloring of the graph with two colors: if one colors all nodes in U **blue**, and all nodes in V **red**, each edge has endpoints of differing colors, as is required in the graph coloring problem. In contrast, such a coloring is impossible in the case of a non-bipartite graph, such as a triangle: after one node is colored **blue** and another **red**, the third vertex of the triangle is connected to vertices of both colors, preventing it from being assigned either color.



We want you to create a program that checks whether a non-directed graph is bipartite or not. This means that is it is possible to paint all his vertices only with two colors in such a way that there are no neighbor vertices with the same color.

Input

The input to your program will consist of more than one test case. Each test case will receive the number of vertices n and the number of edges m, followed of m pairs x, y indicating an edge between x and y. Assume that $1 \leq n \leq 10^4$, $0 \leq m \leq 5n$, that vertices are numbered between 0 and $n-1$, $x \neq y$, and that there is only one edge between x and y.

```
2 1
0 1

4 3
1 2
3 2
3 1
```

Output

For each test case, write the word ***yes*** if the graph is bipartite, ***no*** otherwise. The output of the previous example should be:

yes
no

Solution

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

// function that checks whether a graph is bipartite or not.
// @param[in] G the graph to check.
// @param[in] ini initial vertex of the graph.
// @param[out] colors returning vector with the vertexs that have been already checked (with color assigned).
// @return true when graph is bipartite, false otherwise.
//
bool esBipartit(const vector< vector<int> > &G, int ini, vector<int> &colors){
        // stack the first element of the graph and a color is assigned to it.
        //
        colors[ini] = 1;
        queue<int> q;
        q.push(ini);

        // while there are elements in the queue.
        //
        while (!q.empty()){

                // get a vertex from the queue.
                //
                int u = q.front();
                q.pop();

                // look for all the adjacent vertexs to assign them a color.
                //
                for (size_t i = 0; i < G[u].size(); ++i){

                        // if it has no color assigned then we set it to the opposite color and it is set to the
queue.
                        //
                        if (colors[G[u][i]] == -1){
                                colors[G[u][i]] = 1 - colors[u];
                                q.push(G[u][i]);
                        }
                        // if an adjacent vertex has the same color then the graph is not bipartite.
                        //
                        else if (colors[G[u][i]] == colors[u])
                                return false;
                }
        }
        // if all the connected vertexs are of opposite color  es bipartit.
        //
        return true;
}

int main(){
        int n, m;
        while (cin >> n >> m){

                // read the graph.
                //
                vector< vector<int> > G(n);
                for (int i = 0; i < m; ++i){
                        int x, y;
                        cin >> x >> y;
```

```
                        G[x].push_back(y);
                        G[y].push_back(x);
                }

                // check that all the subgraphs are bipartites.
                //
                bool bipartit = true;
                vector<int> colors(G.size(), -1);
                for (size_t i = 0; i < colors.size() && bipartit; ++i){
                        if (colors[i] == -1){
                                if (!esBipartit(G, i, colors)){
                                        bipartit = false;
                                        break;
                                }
                        }
                }
                if (bipartit) cout << "yes" << endl;
                else cout << "no" << endl;
        }
}
```


CodeWars 2017
Barcelona