



**CodeWars 2018**  
**Barcelona**

**Problems**



#	Problem	Points
1	Savings for the Fallas	1
2	Printing books	1
3	To be able or not to be able. That is the question.	2
4	How old are you?	2
5	Storytelling	2
6	It's the final countdown	3
7	Which graphics card should I buy?	3
8	Run, Forrest, run!	3
9	3D Coffee	4
10	Triathlon timing	4
11	Character counting	4
12	Tracking student's progress	5
13	Quadratic equation solver	5
14	Power of 2	6
15	Not-allowed-entry-charset	6
16	Emirp numbers	7
17	Magic sum	10
18	Space battleship	10
19	Cross-stitch	12
20	Car race	12
21	Game of life	15
22	Nested triangles	15
23	Matrix Code	15
24	Ancient formulas	15
25	Secret door	15
26	Skiing	20
27	Digital castellers	20
28	Meowy's Island	20
29	Hexagons	23
30	Maze	23
31	Keyboard	30



# 1 Savings for the Fallas

1 point

## Introduction

Ana wants to know how much money she will spend to travel from Barcelona to Valencia to enjoy the “Fallas”. She only has a few days to continue saving money to go. Ana will send you a list with the prices of the train tickets. The first line will be the one-way ticket and the second will be the return ticket. Please, can you tell her how many euros she should save?

Come on, help Ana!

## Input

The input consists of two integers in two lines:

<One-way ticket cost>

<Return ticket cost>

## Output

Print out the total of euros that should save following this output format:

Ana should save a total of <total euros> euros.

## Example

### Input

50

35

### Output

Ana should save a total of 85 euros.

## Solutions

### Python3

```
a=int(input())
b=int(input())
print("Ana should save a total of " + str(a+b) + " euros.")
```

## C++

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

int main(){
    int oneWayTicket, returnTicket;
    cin >> oneWayTicket >> returnTicket;
    cout << "Ana should save a total of "
         << (oneWayTicket+returnTicket)
         << " euros." << endl;
}
```

## 2 Printing books

1 point

### Introduction

To earn some extra money, you decide to start working in a printing house. In your first day, you realize the boss looks worried. He should print lots of copies of a book (it's a best-seller!) and he doesn't know how many ink cartridges will remain after printing all the copies. The boss knows that he needs 3 ink cartridges for each book and he has written in a paper the number of books to print and the number of cartridges stored in the warehouse. Using your programming skills, try to help the boss ensure that there are always enough ink cartridges in the warehouse to print all the books.



**HINT:** Consider that there will be always enough ink cartridges in the warehouse.

### Input

The input consists of two integers in two lines:

<Number of books to print>

<Number of ink cartridges available in the warehouse>

### Output

Print the number of remaining cartridges after printing all the books.

### Example

#### Input

10

80

#### Output

50

### Solutions

#### Python3

```
cars_per_book=3
books=int(input())
ink_carts=int(input())
print(ink_carts - books * cars_per_book)
```

**C++**

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

int main(){
    int booksToPrint, inkSupplies;
    cin >> booksToPrint >> inkSupplies;
    cout << (inkSupplies - (booksToPrint*3)) << endl;
}
```



**3**

## To be able or not to be able. That is the question.

*2 points*

### Introduction

Ana wants to buy a new house but she doesn't know if she has enough money. She will tell you how many euros she has saved, and how many euros the house costs. Can you help her to decide if she has enough money to buy the new house?

Come on, help again Ana!

### Input

The input consists of two integers in two lines:

<Ana's savings>

<Cost of the new house>

### Output

Print out the one of the following outputs once you confirm Ana is able or not to buy a new house:

Ana can buy the house !!

Ana can NOT buy the house :(

### Example 1

#### Input

500000

300000

#### Output

Ana can buy the house !!

### Example 2

#### Input

50000

300000

#### Output

Ana can NOT buy the house :(

## Solutions

### Python3

```
savings = int(input())
cost = int(input())
if(savings >= cost):
    print("Ana can buy the house !!")
else:
    print("Ana can NOT buy the house :(")
```

### C++

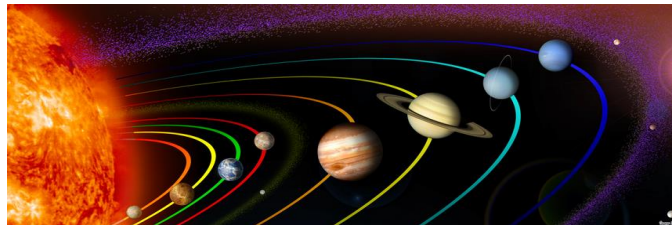
```
#include <iostream>
using namespace std;

int main(){
    int totalSavings, totalCost;
    cin >> totalSavings >> totalCost;
    if (totalSavings >= totalCost)
        cout << "Ana can buy the house !!" << endl;
    else
        cout << "Ana can NOT buy the house :(" << endl;
}
```

## 4 How old are you? *2 points*

### Introduction

The actual definition of a year is the time it takes to a planet to complete a single orbit around the Sun. That is, here on Earth, we consider a year to be 365 days.



But if you were to live in another planet of our Solar System – a year would work out to something else. Fortunately, we have a simple table to get the equivalence in Earth days of any planet's year.

Planet	Revolution period in Earth days
Mercury	88
Venus	225
Earth	365
Mars	687
Jupiter	4333
Saturn	10759
Uranus	30689
Neptune	60182

Since we plan to travel around the Solar System it is important to have conversion software to know the Earth age of the interplanetary travelers. Can you provide an easy way to convert the age expressed in years in a given planet to its value in Earth years?

### Input

The input consists of two lines. The first line is an integer indicating your current age in years in the planet where you live. The second line is a string with the name of the planet where you live.

<Current age in years in the planet where you live>

<Name of the planet where you live>

### Output

Print out your age in Earth years as an integer value.

### Example

#### Input

10  
Mars

#### Output

18

## Solutions

### Python3

```
list = dict([("Mercury",88),      ("Venus",225),
             ("Earth",365),      ("Mars",687),
             ("Jupiter",4333),   ("Saturn",10759),
             ("Uranus",30689),   ("Neptune",60182)])

age = int(input())
planet = input()

earthAge = (age * list[planet])//365

print(earthAge)
```

### C++

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

map<string, int> planet_days = {
    {"Mercury", 88}, {"Venus", 225},
    {"Earth", 365}, {"Mars", 687},
    {"Jupiter", 4333}, {"Saturn", 10759},
    {"Uranus", 30689}, {"Neptune", 60182}
};

int main(){
    int age;
    string planet;
    cin >> age >> planet;
    int total_days = planet_days[planet]*age;
    int earth_age = total_days/planet_days["Earth"];
    cout << earth_age << endl;
}
```

## 5 Storytelling

2 points

### Introduction

A storyteller has been told that nowadays the stories must be dynamic, so he wants to create a program that everybody could use. This program would have a static storyline and a dynamic part, where the reader would introduce his personal information: name, age, gender, city, favourite sport, favourite team and its ideal job.



---

**HINT: Beware of the gender (boy/girl) and the consequent changes derived.**

---

This is the static storyline:

*Name* is a *age* year-old *gender*. *Pronoun* is living with *possesive pronoun* parents in an apartment in the centre of *city*, where *pronoun* hangs out with *possesive pronoun* friends. Moreover, in *possesive pronoun* free time *pronoun* plays *favourite sport* in a team called *favourite team*. *name* would like to pursue a career in *ideal job* when *pronoun* is older, that's why *pronoun* is studying hard.

### Input

The input will be the dynamic data provided in seven lines.

### Output

Print out the whole story including the static storyline part filled with the provided dynamic data.

### Example

#### Input

```
Ainhoa
22
girl
Donostia
basketball
Mundarro
social working
```

#### Output

```
Ainhoa is a 22 year-old girl. She is living with her parents in an apartment in the
centre of Donostia, where she hangs out with her friends. Moreover, in her free time
she plays basketball in a team called Mundarro. Ainhoa would like to pursue a career
in social working when she is older, that's why she is studying hard.
```

## Solutions

### Python3

```
name=input()
age=input()
gender=input()
city=input()
sport=input()
team=input()
job=input()

if "girl" in gender:
    Pronoun="She"
    pronoun="she"
    possessive_pronoun="her"
else:
    Pronoun="He"
    pronoun="he"
    possessive_pronoun="his"

output_string = name + " is a " +age+" year-old "+gender\
+ ". "+Pronoun+" is living with "+possessive_pronoun\
+ " parents in an apartment in the centre of "+city\
+ ", where "+pronoun+" hangs out with "+possessive_pronoun\
+ " friends. Moreover, in "+possessive_pronoun+" free time "\
+pronoun+" plays "+sport+" in a team called "+team\
+ ". "+name+" would like to pursue a career in "+job\
+ " when "+pronoun+" is older, that's why "+pronoun\
+ " is studying hard."

print(output_string)
```

C++

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

int main() {
    string name, age, gender, city, sport, team, job;
    //We need to use getline because the name, city...
    // might be split in two strings
    getline(cin, name); getline(cin, age);
    getline(cin, gender); getline(cin, city);
    getline(cin, sport); getline(cin, team);
    getline(cin, job);
    string pronoun, capitalizedPronoun, possessivePronoun;
    if (gender == "boy") {
        pronoun = "he";
        capitalizedPronoun = "He";
        possessivePronoun = "his";
    }
    else {
        pronoun = "she";
        capitalizedPronoun = "She";
        possessivePronoun = "her";
    }

    cout << name << " is a " << age << " year-old "
        << gender << ". " << capitalizedPronoun
        << " is living with " << possessivePronoun
        << " parents in an apartment in the centre of "
        << city << ", where " << pronoun
        << " hangs out with " << possessivePronoun
        << " friends. Moreover, in " << possessivePronoun
        << " free time " << pronoun << " plays " << sport
        << " in a team called " << team << ". " << name
        << " would like to pursue a career in " << job
        << " when " << pronoun << " is older, that's why "
        << pronoun << " is studying hard." << endl;
}
```

## 6 It's the final countdown

3 points

### Introduction

Different space agencies around the world plan to send manned missions to Mars during 21<sup>st</sup> century.



Many challenges are involved in these projects, like economic funding and astronaut health risks. Some scientists are currently developing and testing the different technologies to provide the first step towards Mars colonization.

And here is a critical part -- we need you to develop the software to generate the final countdown for the spaceship launch. Would you be able to do it?

### Input

The input will be a single positive integer number indicating the first value of the countdown.

### Output

Print out the countdown, that is the sequence backward counting starting with the input value.

### Example

#### Input

3

#### Output

3 2 1 0



## Solutions

### Python3

```
last_number=int(input())
for x in range(last_number, -1, -1):
    print (x, end=" ")
```

### C++

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

int main()
{
    int startNumber;
    cin >> startNumber;
    for (int i = startNumber; i > 0; i--)
        cout << i << " ";
    cout << 0 << endl;
}
```

## 7 Which graphics card should I buy?

3 points

### Introduction

Your old GPU (Graphics Processing Unit) can no longer support newer video games and you want to study several options to check which is the one that will give you those extra fps (frames per second) needed. To evaluate the GPU performance the program will test its frequency versus the minimum required frequency of several video games.

### Input

The input will be a sequence of lines with integer numbers representing frequencies (all in MHz). The first number will be frequency of the GPU to test. The rest of the numbers will be the minimum frequencies required for a specific game title to perform properly. If the game frequency is 0, the process should stop, and not consider that value.

### Output

The output will be the number of video games that will run perfect for the selected GPU.

### Example

#### Input

```
1809
1700
1900
1200
0
```

#### Output

```
2
```

## Solutions

### Python3

```
gpu_frequency=int(input())
game_needed_frequency=int(input())
count_of_games=0
while( game_needed_frequency != 0):
    if(game_needed_frequency<=gpu_frequency):
        count_of_games = count_of_games+1
        game_needed_frequency=int(input())

print(count_of_games)
```

### C++

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

int main()
{
    int gpuFreq, currFreq, games;
    games = 0;
    cin >> gpuFreq;
    cin >> currFreq;
    while (currFreq != 0)
    {
        if (gpuFreq >= currFreq)
            games++;
        cin >> currFreq;
    }
    cout << games << endl;
}
```

## 8 Run, Forrest, run!

3 points

### Introduction

Forrest is passionate about running. To have an accurate tracking of all his activity while running he bought a U.S. sport watch. It reports the distance run every day only in miles. Forrest knows that it is recommended that running shoes should be replaced every 622 kilometers and needs a program to sum up all the year running activity to decide whether to replace or not his running shoes. Keep in mind that 1 mile is approximately 1.6 kilometers.

### Input

The input will be a sequence of 365 integer values in a single line representing the miles run daily during the last year.

### Output

The output of the program reports a simple string stating "Yes" or "No" to know if the running trainers must be replaced.

### Example

#### Input

```
16 4 1 15 12 20 14 2 7 10 4 14 5 15 16 21 13 3 16 11 18 17 10 20 2 18 7 12 11 5 10 8
12 1 6 1 6 12 2 10 19 8 14 13 5 6 8 12 17 1 10 4 18 6 3 7 3 1 14 11 3 14 11 13 6 13 10
14 4 11 3 10 17 18 13 11 17 7 11 3 12 4 9 2 5 15 20 20 16 19 20 18 14 8 9 15 18 21 8 3
13 15 20 17 2 12 8 15 8 4 8 10 11 20 15 1 10 5 16 11 19 11 20 6 18 6 13 21 6 8 6 11 14
14 2 14 7 11 9 6 1 7 1 4 16 20 12 15 4 5 2 20 5 17 15 13 18 18 10 17 7 14 21 19 13 17
2 1 10 11 1 5 19 6 2 12 6 14 6 16 16 15 15 11 10 21 10 11 1 21 12 3 18 20 2 9 20 20 18
5 12 13 17 9 12 1 1 18 7 15 5 21 13 20 16 2 9 10 9 1 3 8 15 16 6 14 15 1 2 9 18 18 2 4
16 14 16 2 1 21 4 3 15 16 16 3 20 6 21 5 1 20 14 4 14 14 7 2 14 9 17 14 20 21 21 19 16
20 11 18 11 5 11 21 6 16 7 18 11 9 20 2 9 12 7 5 14 14 12 15 1 3 8 13 11 20 7 5 9 4 3
15 1 1 19 11 15 12 15 21 10 11 19 19 18 15 3 6 4 20 19 6 21 17 7 2 18 4 19 8 14 16 6
13 3 15 2 12 3 8 4 17 6 7 8 11 14 16 21 20 4 15 5 14 13 3 1 21 2 13 8 4 6 8 10
```

#### Output

Yes

### Solutions

#### Python3

```
import sys
sum = 0
for line in sys.stdin:
    data = line.split()
    for i in data:
        sum = sum + (int(i)*1.6)
    if sum >= 622:
        print("Yes")
    else:
        print("No")
```

**C++**

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

int main()
{
    double suma = 0;
    double num;
    while(cin >> num){
        suma += num*1.6;
    }
    (suma >= 622) ? (cout << "Yes") : (cout << "No");
    cout << endl;
}
```

## 9 3D Coffee

4 points

### Introduction

You have just bought a 3D printer and you want to run a successful 3D coffee bar business. To calculate the price of every printed part you sell, you just apply the following formula:

$$\text{price} = \text{volume printed [cm}^3\text{]} \times p \text{ [€/cm}^3\text{]}$$

Where  $p$  is the price per  $\text{cm}^3$ . If the customer doesn't have a drink in the coffee bar the value of  $p$  is 2 €/cm<sup>3</sup>. But you promote a special offer that if customer buys one or more drinks then the value of  $p$  is reduced to 1.8 €/cm<sup>3</sup>.

So to compute the final price for each printer part your cash register will receive two values: one specifying the volume of the printed part and a second one detailing if a customer has had at least one drink.

### Input

The input consists of several lines of two values following this format:

<3D volume printed with two decimals> <Just Y or N to report if the customer has had a drink>

The input ends with a line with -1 integer value.

### Output

Print out the invoice price for each 3D volume printed with two decimals rounding.

### Example

#### Input

50.12 Y

50.12 N

100 Y

-1

#### Output

90.22

100.24

180.00

## Solutions

### Python3

```
# Read an unknown number or lines until -1
bEOF = 0
input_clients_list=[]
while(bEOF == 0):
    line=input()
    if line=="-1":
        bEOF=1
    else:
        input_clients_list.append(line)

for client in input_clients_list:
    cm3,coffee=client.split()
    if coffee=="Y":
        p=1.8
    else:
        p=2.0
    cost=p*float(cm3)
    rounded_cost=round(cost,2)
    # we want 2 output digits
    print ( "%.2f" % rounded_cost )
```

## C++

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <stdint.h>
using namespace std;

int main()
{
    string line;
    while(getline(cin, line)){
        if (line == "-1") break;
        else{
            string hadDrink;
            double volume, totalAmount, cost;
            istringstream(line) >> volume >> hadDrink;
            if( hadDrink == "Y" )
                cost = 1.8;
            else
                cost = 2;
            totalAmount = volume * cost;
            cout << setprecision(2) << fixed << totalAmount
                << endl;
        }
    }
}
```



# 10 Triathlon timing

4 points

## Introduction

A triathlon is a multidisciplinary race that combines three different sports: swimming, cycling, and running. Although the three disciplines are practiced one after the other, the classificatory system tracks the time of each of them individually. At the end of the race, the three registered times are added up to determine the final time.

Given the three registered times for swimming, cycling and running, compute the final race time in the appropriate format (XXhYYmZZs). The triathlon will not last longer than 72 hours.

## Input

The three registered times for swimming, cycling and running of triathlon are provided following the time format XXhYYmZZs.

## Output

The final race time of triathlon in format XXhYYmZZs.

## Example

### Input

00h28m43s

01h02m31s

00h37m17s

### Output

02h08m31s

## Solutions

### Python3

```
import re
times=[]

for num_times in range(3):
    # No error detection in the input format!
    a = [int(i) for i in re.findall('[0-9]+',input())]
    times.append(a)

# Lets add the times
total=[0,0,0]
# loop hh mm ss
for x in range(3):
    for y in range(len(times)):
        total[x]=total[x]+times[y][x]

# And reduce the minutes and seconds to 60
total[1]=total[1]+int(total[2]/60)
total[2]=total[2]%60

total[0]=total[0]+int(total[1]/60)
total[1]=total[1]%60

print("%02dh%02dm%02ds" % (total[0], total[1], total[2]))
```

## C++

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <stdint.h>
using namespace std;

int main()
{
    string swimLine, cycleLine, runLine;
    cin >> swimLine >> cycleLine >> runLine;

    int secSwim, minSwim, hourSwim;
    secSwim = (int)(swimLine[6] - '0') * 10 + (int)(swimLine[7] - '0');
    minSwim = (int)(swimLine[3] - '0') * 10 + (int)(swimLine[4] - '0');
    hourSwim = (int)(swimLine[0] - '0') * 10 +
                (int)(swimLine[1] - '0');

    int secCycle, minCycle, hourCycle;
    secCycle = (int)(cycleLine[6] - '0') * 10 +
                (int)(cycleLine[7] - '0');
    minCycle = (int)(cycleLine[3] - '0') * 10 +
                (int)(cycleLine[4] - '0');
    hourCycle = (int)(cycleLine[0] - '0') * 10 +
                (int)(cycleLine[1] - '0');

    int secRun, minRun, hourRun;

    secRun = (int)(runLine[6] - '0') * 10 + (int)(runLine[7] - '0');
    minRun = (int)(runLine[3] - '0') * 10 + (int)(runLine[4] - '0');
    hourRun = (int)(runLine[0] - '0') * 10 + (int)(runLine[1] - '0');

    int seconds = secSwim + secCycle + secRun;
    int minutes = minSwim + minCycle + minRun;
    int hours = hourSwim + hourCycle + hourRun;

    minutes += seconds / 60;
    seconds = seconds % 60;
    hours += minutes / 60;
    minutes = minutes % 60;

    if (hours < 10)
        cout << "0" << hours << "h";
    else
        cout << hours << "h";
    if (minutes < 10)
        cout << "0" << minutes << "m";
    else
        cout << minutes << "m";
    if (seconds < 10)
        cout << "0" << seconds << "s";
    else
        cout << seconds << "s";
    cout << endl;
}
```

# 11 Character counting

4 points

## Introduction

Your Computer Science teacher needs a program to analyze text. He is interested in counting how many uppercase and lowercase letters the text contains. The rest of the characters in the text (numbers, punctuation, math symbols, etc.) must be also counted except spacing characters that must be ignored. Would you be so kind to write this program?

## Input

The input will be a line with the text to be analyzed.

## Output

Print out the number of uppercase, lowercase and other characters found in the text.

## Example

### Input

Hi there! How are you today? The answer is  $4*8 + 34 - 2 = (2^6)$

### Output

Uppercase 3  
Lowercase 29  
Other 16

## Solutions

### Python3

```
input_str=input()
up_count=0
low_count=0
other_count=0

for x in input_str:
    if x.isupper():
        up_count+=1
    elif x.islower():
        low_count+=1
    elif x != ' ':
        other_count+=1

print("Uppercase",up_count)
print("Lowercase",low_count)
print("Other",other_count)
```

### C++

```
#include <iostream>
#include <ctype.h>
using namespace std;

int main()
{
    int upperCase = 0;
    int lowerCase = 0;
    int other = 0;
    char c;
    while(cin >> c)
    {
        if (isupper(c))
            upperCase++;
        else if (islower(c))
            lowerCase++;
        else if (c != ' ')
            other++;
    }
    cout << "Uppercase " << upperCase << endl;
    cout << "Lowercase " << lowerCase << endl;
    cout << "Other " << other << endl;
}
```

## 12 Tracking student's progress

5 points

### Introduction

Every quarter your teacher collects the student's notes of the every class where he teaches. He reviews the results and compare the different classes he is teaching. This way he can improve his lessons. To avoid doing this task manually with a pocket calculator you can help him by creating a program. This program should find out the number of students that is below the average grade. Beware that the student's notes can be written with decimal number, as in real life.

### Input

The input consists of several lines containing the student's notes of the whole class expressed in decimal format.

### Output

Print out the number of students below the average.

### Example

#### Input

```
8
0
4.3
6.5
4.7
```

#### Output

```
2
```

## Solutions

### Python3

```
bEOF = 0
grades_list=[]
grades_avg=0
while( bEOF == 0):
    line=""
    # The try/except will allow us reading a input file
    # without an empty final line
    try:
        line=input()
    except:
        bEOF=1
    if line=="":
        bEOF=1
    else:
        grades_list.append(float(line))
        grades_avg+=float(line)

grades_avg=grades_avg/len(grades_list)

# Checks how many students have a grade below the average
count_below=0
for x in grades_list:
    if x < grades_avg:
        count_below+=1

print(count_below)
```

## C++

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

int main()
{
    double avg = 0;
    vector<double> v;
    double grade;
    while(cin >> grade)
    {
        v.push_back(grade);
        avg +=grade;
    }

    avg = avg / v.size();
    int count = 0;

    for (int i = 0; i < v.size(); ++i)
        if( v[i] < avg ) ++count;

    cout << count << endl;
}
```



# 13 Quadratic equation solver

5 points

## Introduction

Jeremy is nervous. He has his first Quadratic Equation exam, and he is about to begin. But Jeremy is the best young programmer of its classroom, and he wonders if he could programmatically solve any Quadratic Equation in the world with a simple code.

Jeremy knows very well that a generic Quadratic Equation has the form

$$ax^2 + bx + c = 0$$

Where  $a$ ,  $b$ , and  $c$  are the coefficients of the equation. Could you help Jeremy and write a program that receives as input the coefficients of the equations and tells you the solution?

Remember that the roots of a Quadratic Equation can be computed as

$$x_{+,-} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

## Input

The input of the program is the value of the coefficients of the equation, separated by spacing characters.

## Output

The output of the program is the solution (the roots of the quadratic equation).

Remember that a quadratic equation can have complex roots when  $b^2 - 4ac$  is lesser than 0. Your program must answer "It has complex Roots!" if that is the case (Example 1).

When the roots are not complex, the solution must be shown featuring first  $x_+$  and second  $x_-$  with two decimals rounding (Example 2).

## Example 1

### Input

1 1 5

### Output

It has complex Roots!

## Example 2

### Input

1 2.5 -4.5

### Output

$x_+ = 1.21$ ;  $x_- = -3.71$

## Solutions

## Python3

```
import math
[a,b,c] = [float(x) for x in input().split()]
# Calculate the square root argument
d = b*b-4*a*c
# Complex roots?
if d < 0:
    print("It has complex Roots!")
else:
    # Else, print the output
    e = math.sqrt(d)
    x1=(-b+e)/(2*a)
    x2=(-b-e)/(2*a)
    print("x_+ = %.2f; x_- = %.2f" % (x1,x2))
```

## C++

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
    double a, b, c;
    cin >> a >> b >> c;
    double x1, x2;
    double check = b*b - 4*a*c;

    if (check < 0)
        cout << "It has complex Roots!" << endl;

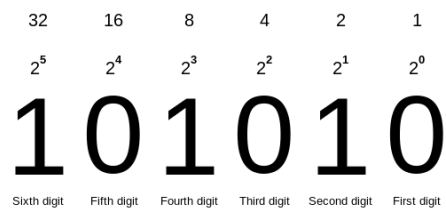
    else
    {
        x1 = (-b + sqrt(check))/(2*a);
        x2 = (-b - sqrt(check))/(2*a);
        cout << "x_+ = " << setprecision(2)
             << fixed << x1 << "; ";
        cout << "x_- = " << setprecision(2)
             << fixed << x2 << endl;
    }
}
```

# 14 Power of 2

6 points

## Introduction

In Computer Science we deal daily with binary numbers. As you may know a binary number is represented in a row of bits that could have a value of 0 or 1. It is assigned to each bit a position number, ranging from zero to N-1, where N is the number of bits in the binary representation used. Usually, this is simply the exponent for the corresponding bit weight in base-2 (such as in  $2^{31}..2^0$ ).



Consider the decimal number 42, as an example, its representation in binary is 101010. Meaning that the position of  $2^5$  is 1,  $2^4$  is 0,  $2^3$  is 1,  $2^2$  is 0,  $2^1$  is 1 and  $2^0$  is 0. So, the exponent of the highest power of 2 is 5 ( $2^5$ ).

Given a decimal number find out the exponent of the highest power of 2.

## Input

The input will be a single positive integer number higher than 0.

## Output

Print out the position of first one from left in binary.

### Example 1

#### Input

5

#### Output

2

### Example 2

#### Input

10

#### Output

3



## Solutions

### Python3

```
num = int(input())
# Converts to a binary string
num_bin=bin(num)
# And check the max power. I.e 42=0b101010 -> Max power==5
print( len(num_bin)-2-1)
```

### C++

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
    int num;
    double res;
    cin >> num;
    res = log2((double)num);
    cout << (int) res << endl;
}
```

# 15 Not-allowed-entry-charset

6 points

## Introduction

An online Web form includes a section for entering freeform text. However, there are some characters or numbers that cannot be inserted into our back-end database. To solve this problem, we need you to create a program that, given a string and a not-allowed character set, will remove all occurrences of the not-allowed literal characters from the entry of the user string.

## Input

The input will consist of two lines:

<the given string>

<the not-allowed-entry-char-set>

## Output

Print out the original given string after removing the not-allowed-entry-charset.

## Example 1

### Input

12345678

3456

### Output

1278

## Example 2

### Input

87654321

346

### Output

87521

## Solutions

### Python3

```
# Read the input
string = input()
not_allowed = input()
output_string=""
# Creates the output by adding the allowed characters
for x in string:
    if not(x in not_allowed):
        output_string+=x
print( output_string)
```

### C++

```
#include <iostream>
#include <string>
#include <stdint.h>
using namespace std;

int main()
{
    string original;
    string nA;
    string result;

    getline(cin, original);
    cin >> nA;

    for (int i = 0; i < original.length(); i++)
    {
        size_t found = nA.find(original[i]);
        // If not found then append this char to the result
        if (found==string::npos)
        {
            result+= original[i];
        }
    }

    cout << result << endl;
}
```

# 16 Emirp numbers

7 points

## Introduction

A prime is a number that is only divisible by one and itself, which is essentially saying that it has no divisor. Nowadays primes are essential for secure communications. Most modern computer cryptography works by using the prime factors of large numbers.

When you reverse the digits of most primes you get a composite number (for example, 43 becomes 34). That is not the case for palindromic primes that read the same forward and backward (for example, 727), so reversing a palindromic prime gives you the same prime.

Then there is an special category, the emirp numbers. An emirp (the word "prime" written backwards) is a prime whose reversal is also prime, but which is not a palindromic prime (for example, 13 becomes 31). The first emirp numbers are 13, 17, 31, 37, 71, 73, 79, 97, 107, 113, 149, 157, ...

Let's write a program to find out whether an integer number is emirp.

## Input

The input will be an integer number.

## Output

Print out whether the given number is emirp or not.

## Example 1

### Input

13

### Output

13 is an emirp number

## Example 2

### Input

11

### Output

11 is not an emirp number

## Solutions

## Python3

```
def isPrime(x):
    prime = True
    i = 2
    while i < x:
        if (x % i == 0):
            prime = False
            break
        i += 1
    return prime

def revertNumber(x):
    reverse = 0
    while x > 0:
        remainder = x % 10
        reverse = (reverse * 10) + remainder
        x = x // 10
    return reverse

def isPalindrome(x):
    return (str(x) == str(x)[::-1])

number = int(input())

result = isPrime(number) and isPrime(revertNumber(number))
result = result and not isPalindrome(number)

if result:
    print(str(number) + " is an emirp number")
else:
    print(str(number) + " is not an emirp number")
```



## C++

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

bool isPrime(int n)
{
    if (n <= 1) return false;
    for (int i = 2; i < n; i++)
        if (n % i == 0) return false;
    return true;
}

bool isEmirp(int n)
{
    if (!isPrime(n)) return false;
    int rev = 0, ori = n;
    while (n != 0)
    {
        int d = n % 10;
        rev = rev * 10 + d;
        n /= 10;
    }
    //check that the reverse is prime and not a plaindrome
    return isPrime(rev) && !(ori == rev);
}

int main()
{
    int n;
    cin >> n;
    cout << n << " is ";
    if (!isEmirp(n)) cout << "not ";
    cout << "an emirp number" << endl;
}
```

# 17 Magic sum

10 points

## Introduction

One day, while you are sitting in front of your computer, something incredible happens. The computer starts making some noise and in a blast of light everything goes blank. When you opened your eyes, you found yourself in a weird and magical world. The computer must have sent you to another planet in a different parallel universe.

Walking around, in a hurry to find a way to come back home, you found out a message in a poster saying: "Solve the sum of all digits of a given number to go home!". You stopped and told yourself - wait, to come back home, the only thing is to solve a simple sum? is all that takes! –

Unfortunately, in this world, the basic mathematical operations are quite different than the ones in your home world. Thus, you need to understand them in order to give the correct answer and come back home.

The magic sum operation requires you to review a sequence of digits and find the sum of all digits with the following properties:

- In order to consider a digit, it has to match the next digit in the sequence. The sequence is circular, so the digit after the last digit is the first digit in the sequence.
- If a digit is even, you must multiply it by its position in the list before sum it. In this case, the first digit is considered to have the position 1.
- If a digit is odd, just sum it.

For instance:

- 1122 : outputs 7 =  $1+6$  → because the first digit (1) matches the second digit and its odd, we sum its value. The third digit (2) matches the fourth one and because its even, we multiply it by its position (that is 3).
- 123 : outputs 0 → because none of the elements matches the next in the sequence.
- 565 : outputs 5 → because the the third digit (5) matches the first one and its odd.

Will you be able to solve the sum and come back home?

## Input

The input consists of several lines with given numbers

Each number consists in a sequence of digits. All digits are between 1 and 9.

## Output

The output must consist in several lines with the integers representing the magic sum of the input numbers.

## Example

### Input

1122  
123  
565  
1

### Output

7  
0  
5  
1



## Solutions

### Python3

```
# Read the input
bEOF = 0
numbers_list=[]

while( bEOF == 0):
    line=""
    # The try/except will allow us reading a input file
    # without an empty final line
    try:
        line=input()
    except:
        bEOF=1
    if line=="":
        bEOF=1
    else:
        numbers_list.append(line)

for number in numbers_list:
    # to make the circular check easier adds the first
    # digit to the end of the string
    number+=number[0]

    sum=0
    for index in range(len(number)-1):
        # Check if the digit is equal to the next one
        digit=int(number[index])
        next_digit=int(number[index+1])
        if digit==next_digit:
            # and if are equal, checks if it's even or odd
            if digit%2==0:
                sum+=(index+1)*digit
            else:
                sum+=digit
    print(sum)
```

C++

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

// return the magic sum for a given number and
//its next in the sequence
int sum(int x1, int x2, int pos) {
    int s = 0;
    if (x1 == x2) {
        // is number even
        if ((x1 & 1) == 0) s = x1 * pos;
        else s = x1;
    }
    return s;
}

// calculates the magic sum of a given sequence of
numbers
int magicSum(vector<int> seq) {
    // perform calculations
    int s = 0;
    int i;
    for (i = 0; i < seq.size() - 1; i++) {
        s += sum(seq[i], seq[i + 1], i + 1);
    }
    s += sum(seq.back(), seq[0], i + 1);
    return s;
}
//continues on next page...
```

```
//...
// convert a string into an array of integers
vector<int> toVectorInt(string seq) {
    vector<int> seq_i;
    seq_i.reserve(seq.size());
    for (string::iterator it = seq.begin();
         it != seq.end(); ++it) {

        seq_i.push_back((int) (*it - '0'));
    }
    return seq_i;
}

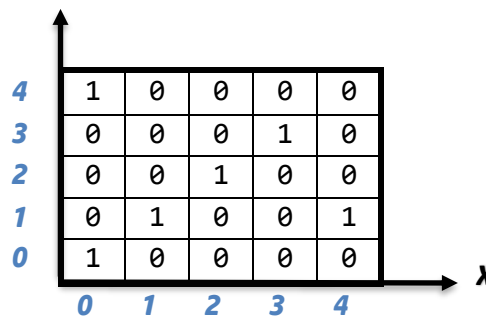
int main() {
    string seq;
    vector<int> seq_i;
    int s;
    // read a sequence --> a line
    while (cin >> seq) {
        //transform the string into a integer vec-
tor
        seq_i = toVectorInt(seq);
        //compute the sum
        s = magicSum(seq_i);
        //write the output
        cout << s << endl;
    }
}
```

# 18 Space battleship

10 points

## Introduction

A long time ago in a galaxy far, far away.... the Rebel and Imperial fleets are fighting against each other in the final battle near Coruscant. Princess Leia has ordered you to provide the total amount of hits against the Imperial forces. To ease your job you are given a square matrix representing the battle scenario with the X/Y coordinates where the imperial ships are present. You are also provided with the X/Y coordinates of the shots fired by the rebels.



Sometimes the rebels fire more than once to the same objective. Beware of not double counting these shots. Can you develop a program to provide this data as soon as possible? Remember you are her last hope.

## Input

The first value of the input will be an integer number representing the size of the square matrix. It will be followed by the matrix where 1 means an imperial spaceship is present and 0 is just empty space.

Then the number of shots is reported followed by the coordinates of every shot.

## Output

Print out the number of successful hits on the Imperial fleet.

## Example

### Input

```
5
1 0 0 0 0
0 0 0 1 0
0 0 1 0 0
0 1 0 0 1
1 0 0 0 0
5
1 1
4 4
1 1
4 1
0 4
```

### Output

```
3
```

## Solutions

### Python3

```
# Read the input
# First, fill the battleships matrix
# Beaware of the revesed notation in the matrix
# and the input file ;-)
n = int(input())
# empty matrix
matrix = [[None for i in range(n)] for x in range(n)]

for line in range(n):
    line_int=[int(x) for x in input().split()]
    # and stores it, also in inverted order
    matrix[n-line-1]=line_int

# Read the shots' coordinates and check if it's a hit
num_shots=int(input())
num_hits=0

for trials in range(num_shots):
    [x,y]=[int(i) for i in input().split()]
    if matrix[y][x]==1:
        num_hits+=1
        matrix[y][x]=0

print(num_hits)
```



## C++

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

using namespace std;

int main()
{
    // Read the matrix size
    int n;
    cin >> n;
    // Read the matrix data
    vector < vector <int> > matrix(n, vector<int>(n));
    for (int i=0; i<n; i++){
        for (int j=0; j<n; j++){
            cin >> matrix[j][n-1-i];
        }
    }
    // Read the number of shots
    int shots = 0, counter = 0;
    cin >> shots;
    // Process the shots
    for (int i=0; i<shots; i++)
    {
        int x, y;
        cin >> x >> y;
        if (matrix[x][y] == 1){
            matrix[x][y] = 0;
            counter++;
        }
    }
    cout << counter << endl;
}
```

# 19 Cross-stitch

12 points

## Introduction

As a hobby you decided to do a cross-stitch picture. You have an schema with different colors and you want to know how many skeins of yarn you need for every color. Every stitch consumes 10mm of yarn and every skein has 1m of yarn.

Every character represents an unique color except white spaces and new lines.

## Input

The input will be a line with an integer that represents the total number of colors, followed by a series of lines representing the picture where each different character represents a color.

## Output

A report detailing the number of skeins needed by color. The output order is not relevant.

## Example

### Input

```
3
-----
-----AA-----
-----AAAAAA-----
-----AAAAAAAAAA-----
-----AA-----
-----AAAAAA-----
-----AAAAAAAAAA-----
-----AA-----
-----AAAAAA-----
-----AAAAAAAAAA-----
-----00-----
-----00-----
-----00-----
```

### Output

- 2 skeins of yarn -
- 1 skeins of yarn A
- 1 skeins of yarn 0

## Solutions

### Python3

```
from math import ceil
stitches_per_skein=100

# Read the input
num_colors=int(input()) # really...this is not needed

# Fills the matix
bEOF = 0
picture=[]

while(bEOF == 0):
    line=""
    # The try/except will allow us reading a input file
    # without an empty final line
    try:
        line=input()
    except:
        bEOF=1
    if line=="":
        bEOF=1
    else:
        picture.append(line)

# Create a dynamic dictionary with the
# color symbols as keys and the count as value
dictionary = {}
# Now, loop for all the lines
for line in picture:
    # and look at all the stitches
    for stitch in line:
        if not(stitch in dictionary):
            dictionary[stitch] = 1
        else:
            dictionary[stitch] += 1

# Print the results
for key, value in dictionary.items():
    skeins = str(ceil(value/stitches_per_skein))
    yarn = str(key)
    print("- "+ skeins +" skeins of yarn " + yarn)
```

## C++

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

int main()
{
    map<char, int> m;
    const int mmPerStitch = 10, mmPerSkein = 1000;
    string line;
    int num_colors;
    cin >> num_colors;
    while (getline(cin, line)){
        // Read all the stitches of current line
        for (int i = 0; i < line.length(); ++i){
            char stitch = line[i];
            // Is the color of this stitch already seen?
            map<char, int>::iterator it = m.find(stitch);
            // Increase the counter for current color
            if (it != m.end()) m[stitch] = m[stitch]+1;
            // New color, so let's initialize the counter
            else m[stitch] = 1;
        }
    }

    for (map<char, int>::iterator it = m.begin();
        it != m.end(); ++it){

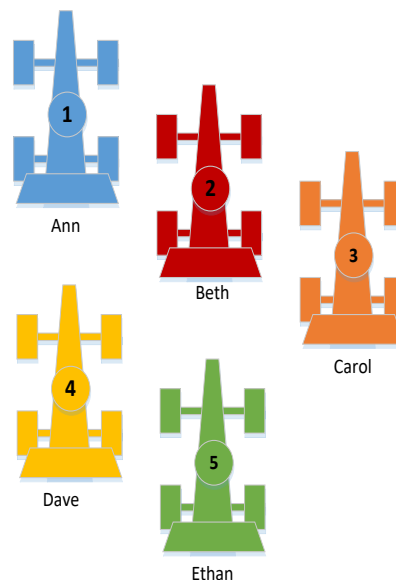
        int skein = (it->second*mmPerStitch)/mmPerSkein;
        int module = (it->second*mmPerStitch)%mmPerSkein;
        cout << "- " << (module == 0? skein : skein+1);
        cout << " skeins of yarn " << it->first << endl;
    }
}
```

# 20 Car race

12 points

## Introduction

A famous website portal has asked you to help them publish the race results of car races.



You have managed to subscribe to a news feed service that reports the order of the racers in the grid at the beginning of the race and any overtakes that occur during the race. This will suffice to let you keep track of each racer's position until the end of the race. Can you automate this task writing a piece of software?

## Input

The input format is as follows:

- The first line consists of the list of racers following the order in the grid. The name of the racers consists of a first name without any white spaces. In the example; Ann is first followed by Beth, which is followed by Carol, etc.
- The following lines contain the overtakes that occur over the course of the race. All the overtake announcements will consist one car taking the position of the car that was immediately ahead of it. Do not consider the case of the racers that are lapped twice by the leading racers.

## Output

The output consists of the list of racers, one per line, in the order in which they have finished the race.

## Example

### Input

```
Ann Beth Carol Dave Ethan  
Ethan overtakes Dave  
Carol overtakes Beth  
Carol overtakes Ann  
Beth overtakes Ann  
Dave overtakes Ethan
```

### Output

```
Carol  
Beth  
Ann  
Dave  
Ethan
```

## Solutions

### Python3

```
# Read the input and creates a vector where the index  
# is the position and the value is the name  
order=input().split()  
# Now, loops though the list of overtakes  
# Fills the matix  
bEOF = 0  
while( bEOF == 0):  
    line=""  
    # The try/except will allow us reading a input file  
    # without an empty final line  
    try:  
        line=input()  
    except:  
        bEOF=1  
    if line=="":  
        bEOF=1  
    else:  
        # Here we have a valid line  
        [racer_2, dummy, racer_1]=line.split()  
        # Looks for the index of the 1st racer  
        # the second racer is its follower  
        racer_1_pos=order.index(racer_1)  
        # And switch the positions  
        tmp_str=order[racer_1_pos]  
        order[racer_1_pos]=order[racer_1_pos+1]  
        order[racer_1_pos+1]=tmp_str  
  
# And just print the final order  
for racer in order:  
    print(racer)
```

**C++**

```

#include <iostream>
#include <string>
#include <sstream>
#include <map>
#include <vector>
using namespace std;

int main(void)
{
    string line, token;
    map<string, int> ranking;
    getline(cin, line);
    istringstream ss(line);
    int i=0;
    while (getline(ss, token, ' '))
        ranking[token]=i++;
    int N = i;
    string carA, overtakes, carB;
    while (getline(cin, line))
    {
        ss.str(line);
        ss.clear();
        ss >> carA >> overtakes >> carB;
        ranking[carA]--; // carA is now one position ahead.
        ranking[carB]++; // carB is now one position after.
    }
    vector<string> v(N);
    for (map<string, int>::iterator it = ranking.begin();
        it!=ranking.end(); it++){

        v[it->second] = it->first;
    }
    for (i=0;i<N;i++)
        cout << v[i] << endl;

    return 0;
}

```



## 21 Game of life

15 points

### Introduction

The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970.

The game is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves.

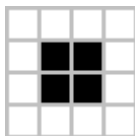
We want to simulate such experiment, by creating a simulator of this Game of Life.

The rules of the game are simple.

We have a board of  $N \times N$  size, that will be filled with life or dead cells (# or .). After each turn, every state of the cell will be determined by the previous state according to the following rules:

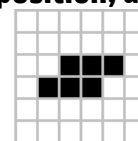
1. Any live cell with fewer than two live neighbors dies, as if caused by under-population.
2. Any live cell with two or three live neighbors' lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by over-population.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

The neighbors of a cell A are determined by all the surroundings cells of the cell A (vertically, horizontally and diagonally). Some patterns will survive forever, other may oscillate among different status and other may even move or create other patterns.

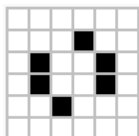


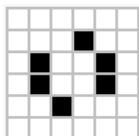
This will be a stable pattern, . As you can see all the dead cells will remain dead as they only have 1 or 2 neighbors, and the already alive cells will always live because they have 3 neighbors.

An oscillator is a pattern that returns to its original state, in the same orientation and position, after a



finite number of turns. The toad is a repeating pattern that oscillates between



. Those cells that appear, do so because they have exactly 3 neighbors, and those cells that die do so because they have 4 or more neighbors or they have 1 or less neighbors.

Following these rules, we can create complex and bigger patterns.

Your goal will be create an application that can load a board of size  $N \times N$  from the input, and will produce as output the final state of the board after  $m$  turns.

You can see some examples of input and output in the following sections.



**HINT: Don't forget that the total size of the board can be up to 100 x 100, so be careful with sizes, and don't forget to be careful when updating your board (wink, wink). By the way, doing it by hand could take LOTS of time (specially if we will test it with high numbers of iterations...).**

### Input

- < m that represents the number of turns >
- < N that represents the board size >
- < board data matrix displayed as N rows x N columns >

### Output

Print out the resulting board data.

#### Example 1

##### Input

```
2
5
. . . . .
. . # # .
. . # # .
. . . . .
. . . . .
```

##### Output

```
. . . . .
. . # # .
. . # # .
. . . . .
. . . . .
```

#### Example 2

##### Input

```
2
5
# . # # .
# . . # .
. . # # .
. . . . .
. . . . .
```

##### Output

```
. . # # .
. # . . #
. . . # .
. . . . .
. . . . .
```



## Solutions

## Python3

```
# Read the input
M=int(input())
N=int(input())
grid=[]
for _ in range(N):
    line=input().split()
    grid.append(line)

# Empty grid
future=[[None for _ in range(N)] for _ in range(N)]

# Turns loop
for turn in range(1, M+1):
    for i in range(N):
        for j in range(N):
            alive = 0
            # Calculate the limits of the adjacent cells
            # taking in consideration the grid edges
            x_min=max(0, i-1); x_max=min(i+1, N-1)
            y_min=max(0, j-1); y_max=min(j+1, N-1)

            for x in range(x_min, x_max+1):
                for y in range(y_min, y_max+1):
                    if(grid[x][y]=='#' and (x!=i or y!=j)):
                        alive+=1

            # Cell is lonely and dies
            if(grid[i][j] == '#' and (alive < 2)):
                future[i][j] = '.'
            # Cell dies due to over population
            elif (grid[i][j] == '#' and (alive > 3)):
                future[i][j] = '.'
            #A new cell is born
            elif (grid[i][j] == '.' and (alive == 3)):
                future[i][j] = '#'
            #Remains the same
            else: future[i][j] = grid[i][j];

        # And copy the new grid to the current one
        for i in range(N):
            for j in range(N):
                grid[i][j]=future[i][j]

# Print the result
for i in range(N):
    for j in range(N):
        print(grid[i][j],end=' ')
    print()
```

## C++

```
#include <iostream>
#include <vector>
using namespace std;
//global variables to simplify parameter passing
int M, N;
bool valid_position(int i, int j);
void read(vector< vector<bool> >& grid);
void print(const vector< vector<bool> >& grid);
void turn(vector<vector<bool> >& grid);
//Program the main structure
int main(){
    cin >> M >> N;
    vector< vector< bool> > grid(N, vector<bool>(N));
    read(grid);
    // Compute each turn
    for (int k = 0; k < M; ++k){
        turn(grid);
    }
    print(grid);
}
//The detailed implementation is in the functions
bool valid_position(int i, int j){
    return(i >= 0 and i < N and j >= 0 and j < N);
}

void read(vector< vector<bool> >& grid){
    for (int i = 0; i < N; ++i){
        for (int j = 0; j < N; ++j){
            char c;
            cin >> c;
            grid[i][j] = (c == '#');
        }
    }
}

void print(const vector< vector<bool> >&grid){
    for (int i = 0; i < N; ++i){
        for (int j = 0; j < N; ++j){
            char c = (grid[i][j]? '#' : '.');
            cout << c;
            if (j!=N-1)
                cout << " ";
        }
        cout << endl;
    }
}
//continues on the next page...
```

```
//...

void turn(vector<vector<bool> >&grid){
    // Loop through every cell
    vector<vector<bool> > future(N, vector<bool>(N));
    for (int l = 0; l < N; l++){
        for (int m = 0; m < N; m++){
            // finding number of alive neighbours
            int alive = 0;
            for (int i = -1; i <= 1; i++){
                for (int j = -1; j <= 1; j++){
                    //don't look at invalid positions
                    //to avoid segmentation faults
                    if(valid_position(l+i, m+j)){
                        alive += grid[l + i][m + j];
                    }

                    // The cell needs to be subtracted from
                    // its neighbours as it was counted before
                    alive -= grid[l][m];
                    // Implementing the Rules of Life
                    // Cell is lonely and dies
                    if ((grid[l][m] == 1) && (alive < 2))
                        future[l][m] = 0;
                    // Cell dies due to over population
                    else if ((grid[l][m] == 1) && (alive > 3))
                        future[l][m] = 0;
                    // A new cell is born
                    else if ((grid[l][m] == 0) && (alive == 3))
                        future[l][m] = 1;
                    // Remains the same
                    else future[l][m] = grid[l][m];
                }
            }
            grid = future;
        }
    }
}
```

## 22 Nested triangles

15 points

### Introduction

Given an input nesting level number, within the range from 0 to 9, draw a set of triangles which sit inside each other. Please note in the examples that a nesting level of 0 means a 0 triangle.

### Input

The input will be a single integer number indicating the level of nesting between 0 and 9.

### Output

Print out the nested triangles.

### Example 1

#### Input

1

#### Output

1  
111

### Example 2

#### Input

3

#### Output

3  
323  
32123  
3211123  
32222223  
3333333333

### Example 3

#### Input

0

#### Output

0

## Solutions

## Python3

```
def nestedTriangle( level):

    listOfRows = []

    if level == 0:
        listOfRows = ["0"]
    elif level == 1:
        listOfRows = ["1", "111"]
    else:
        listTmp = nestedTriangle(level-1)
        listOfRows = [level]
        for i in listTmp:
            listOfRows.append(str(level) + str(i) + str(level))
            listOfRows.append(str(level) * (len(listOfRows[-1])+2))

    return listOfRows

number = input()

l = nestedTriangle(int(number))

if len(l) != 0:
    size = len(l[-1])
    steps = int(size/2)

    for i in l:
        res = ((str(" ") * steps) + str(i))
        steps= steps - 1
        print(res)
```

## C++

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

int main(){
    int n;
    cin >> n;
    int rows = n*2, stop = 0;
    for (int i = 1; i < rows+1; ++i){
        int start = n;
        //spaces
        for (int j = 1; j < rows-i+1; ++j){
            cout << " ";
        }
        //top
        if(i <= rows/2){
            for(int j = rows-i; j < rows; ++j){
                cout << start;
                --start;
            }
            ++start;
            for(int j = rows-i+1; j < rows; ++j){
                ++start;
                cout << start;
            }
        }
        //bottom
        else{
            ++stop;
            for(int j = start; j > stop; --j){
                cout << start;
                --start;
            }
            for(int j = 0; j < stop*4-1; ++j){
                cout << stop;
            }
            for(int j = stop; j < n; ++j){
                ++start;
                cout << start;
            }
        }
        cout << endl;
    }
}
```



## 23

## Matrix Code

15 points

**Introduction**

You work as a technical consultant for the new upcoming sci-fi film “Matrix Code”.

In the movie, a white hat hacker named “Robt” is studying an unknown new malware (malicious software) that is obfuscating all the internet communications. During the investigation Robt gets absorbed into the machine computing world by the malware and he will have to fight it from the inside.

This movie will use a lot of special effects related to how the computers interact with the real world and the director and art team want to include subliminal messages in the representation of the obfuscated transmissions.

You are asked to develop a prototype to allow the movie team to obfuscate text communications. Since the movie will be set in the 1990’s, the output will have to be represented in 4:3 screen format, to simulate the old CRT monitors.

At the moment, only one obfuscating algorithm is requested. This one, should change all the letters from the text by the next one in the alphabet. If the obfuscated text message does not fill the whole screen, the character “ ” should be shown, except in those pixels that are located in the screen’s diagonal, where # should be shown instead.

**Important notes**

- Format 4:3 means, only character displays of 4x3, 8x6, 12x9, 16x12, etc, are allowed
- Only letters should change, any other symbol should remain the same.
- The diagonal is formed by the positions (x, y) where  $x=y$
- Each element should be separated by a space.

**Input**

A text of any length in a single line (only ASCII characters)

**Output**

The output will be the minimum matrix of format 4:3 showing the obfuscated text with the requested restrictions

### Example 1

**Input**

Hi Robt, the matrix has you and you will not escape!

**Output**

```
I j   S p c u ,   u i f
  n b u s j y   i b t
z p v   b o e   z p v
x j m m   o p u   f t d
b q f ! # " " " " " " "
" " " " " # " " " " " "
" " " " " " # " " " " "
" " " " " " " # " " " "
" " " " " " " " # " " " "
```

### Example 2

**Input**

Hi Robt, to be able to exit from the matrix you must analyze the data prompted in the screen. Then you will fully understand the code. Zzz.

**Output**

```
I j   S p c u ,   u p   c f   b
c m f   u p   f y j u   g s p n
  u i f   n b u s j y   z p v
n v t u   b o b m z A f   u i f
  e b u b   q s p n q u f e   j
o   u i f   t d s f f o .   U i
f o   z p v   x j m m   g v m m
z   v o e f s t u b o e   u i f
  d p e f .   a A A . " " " " "
" " " " " " " " " # " " " " "
" " " " " " " " " # " " " " "
" " " " " " " " " # " " " " "
```



## Solutions

## Python3

```
# Takes any message and prints it in a 4:3 matrix.
def paint(message):

    # Decide the matrix size:
    length = len(message) # Minimum required length
    wRes = 4
    hRes = 3
    width = 4
    height = 3

    while((width * height) < length):
        width += wRes
        height += hRes

    # Here is where the message will be printed
    matrix = ''
    position = 0

    for i in range(height):
        for j in range(width):
            # Write the corresponding character in the matrix:
            # Give priority to write the message:
            if(position < length):
                matrix += message[position]
            # When the message has been printed, fill with # or "
            elif(i == j):
                matrix += '#'
            else:
                matrix += ' '
            # Add spaces or line-end if necessary:
            # End of the line but not end of the matrix
            if((j+1 == width) & (position < (width*height-1))):
                matrix += '\n'
            # No end of the line
            else:
                matrix += ' '
                # Increment position counter
                position += 1

    print(matrix)

# continues on next page...
```

```
#...
# Takes the input message and shifts letters
# by their next ones in the alphabet
def translate (inputMessage):

    # After 'z', it comes 'a' again
    alphabetLowerCase = 'abcdefghijklmnopqrstuvwxyz'
    alphabetUpperCase = 'ABCDEFGHIJKLMNPOQRSTUVWXYZ'
    alphabet = alphabetLowerCase + alphabetUpperCase

    outputMessage = ''
    # Iterate over the input message
    for character in inputMessage:
        if character in alphabet:
            # Find the letter and take the next one in the list.
            newCharacter = alphabet[alphabet.index(character)+1]
        else:
            # Copy the character. No conversion here
            newCharacter = character
        # Append it into the output message
        outputMessage += newCharacter
    return outputMessage

inputMessage = input()
print(translate(inputMessage))
```

## C++

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

// Takes any message and prints it in a 4:3 matrix.
void paint(string message){
    // Decide the matrix size:
    int length = message.size(); // Minimum required length
    int wRes = 4, hRes = 3, width = 4, height = 3;
    while((width * height) < length){
        width += wRes;
        height += hRes;
    }

    // Here is where the message will be printed
    string matrix = "";
    int position = 0;

    for (int i = 0; i < height; ++i){
        for (int j = 0; j < width; ++j){
            // Write the corresponding character in the matrix:
            // Give priority to the message:
            if(position < length){
                matrix += message[position];
            }
            // When the message has been printed fill with # or "
            else if(i == j) matrix += '#';
            else matrix += '"';
            // Add spaces or line-end if necessary:
            // End of the line but not end of the matrix
            if((j+1 == width) and (position < (width*height-1))){
                matrix += '\n';
            }
            // No end of the line
            else matrix += ' ';
            // Increment position counter
            position += 1;
        }
    }

    cout << matrix << endl;
}

// continues on next page...
```

```
//...
// Takes the input message and shifts letters
// by their next ones in the alphabet
string translate (string inputMessage){
    // After 'z', it comes 'a' again
    string alphabetLowerCase = "abcdefghijklmnopqrstuvwxyz";
    string alphabetUpperCase = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    string alphabet = alphabetLowerCase + alphabetUpperCase;
    string outputMessage = "";
    // Iterate over the input message
    for (int i = 0; i < inputMessage.size(); ++i){
        char newChar;
        //search the alphabet for the char at position i
        size_t found = alphabet.find(inputMessage[i]);
        if (found != string::npos){
            // Find the letter and take the next one in the list.
            newChar = alphabet[found+1];
        }
        // Copy the character. No conversion here
        else newChar = inputMessage[i];
        // Append it into the output message
        outputMessage += newChar;
    }
    return outputMessage;
}

int main(){
    string inputMessage;
    getline(cin, inputMessage);
    paint(translate(inputMessage));
}
```

## 24 Ancient Formulas

15 points



### Introduction

Archaeologists found ancient mathematical formulas that use parenthesis "()", brackets "[]" and braces "{}" to group sub-expressions in order to provide better clarity than if they only used parenthesis "()".

Here is an example:  $[1 + 1] + (2 + \{1 + [4 * (2 + 1) + 3]\})$

In any case, the meaning of these grouping symbols is equivalent and they simply indicate the opening and closing of a sub-expression group.

Interestingly, we discovered that some of the closing symbols do not always match with the opening symbol type. For instance, in the formula  $2+({3-2]-1)$  the opening symbol '{' should be closed using a matching '}' instead of a ']' for consistency.

You should write a program that reads one of these ancient formulas and adjust any closing symbols to match the corresponding opening symbol type.

### Input

The input is one line with a formula that may contain several grouping symbols. You can assume that the number of opening and closing symbols match and the only problem is with the consistency between the opening type and the corresponding closing type.

### Output

The output corresponds of a first line with the input formula after adjusting any closing symbols that do not match the opening symbol types. And a second line that informs of the number of closing symbols that had to be modified in the original formula to fix it using the format "# fixes made to the formula."; where # is the number of changes.

### Example 1

#### Input

$2+({3-2]-1)$

#### Output

$2+({3-2}-1)$

1 fixes made to the formula.

### Example 2

#### Input

$[\sin(1) + \rho) + \phi (a + \{1 + [4 * (2\pi + 1) + b]\})]$

#### Output

$[\sin(1) + \rho] + \phi (a + \{1 + [4 * (2\pi + 1) + b]\})]$

3 fixes made to the formula.

## Solutions

### Python3

```
opened="{["
closed=")]"

# Read the input
input_formula=input()
correct=""
corrections=0
queue=""

# Loops though the formula chars
for character in input_formula:
    if character in opened:
        # adds opening symbols to the queue
        queue=queue+character
        correct+=character
    elif character in closed:
        # We expect to find a closing symbol equivalent
        # to the last opened.
        # Close with the equivalent of the opening
        # and increase corrections if it was different.
        exp=closed[opened.find(queue[len(queue)-1])]
        correct+=exp
        if exp!=character:
            corrections+=1
        # Remove the last symbol from the queue
        queue=queue[:len(queue)-1]
    else:
        correct+=character

print(correct)
print(corrections,"fixes made to the formula.")
```



## C++

```
#include <iostream>
#include <string>
#include <stack>
using namespace std;

int main(){
    string line;
    stack<char> s;
    int fixes=0;

    getline(cin, line);
    for (int i=0;i<line.size();i++){
        char c = line[i];
        switch (c){
            // Save the expected closing symbol for the
            // group that is opening right now
            case '(':
                s.push(')');
                break;
            case '[':
                s.push(']');
                break;
            case '{':
                s.push('}');
                break;
            // Regenerate any closing symbol with the
            // expected closing symbol regardless of
            // whether the closing symbol was correct
            case ')':
            case ']':
            case '}':
                if (c != s.top()){
                    c = s.top();
                    fixes++;
                }
                s.pop();
                break;
        }
        cout << c;
    }
    cout << endl;
    cout << fixes << " fixes made to the formula." << endl;
}
```

## 25 Secret door

15 points

### Introduction

As every year, our grandpa Santa Claus is in his journey to deliver presents to all good girls and boys around the world in his sleigh led by magical reindeer. But this time, and out of his imagination, the magical sleigh crashed and Santa landed in a maze of rooms created by the Grinch. By the time Santa recovered from this problem, there are only a few minutes until midnight and Santa is seeking help from a good girl or boy to help him out.

The maze is as follows: every room is connected, by a secret hall, to another room and there is only one room with an exit to the exterior, marked with a 0 (zero).

When you are in one room, you take the secret hall to another room. From there you walk to other room, and so on until you find the exit. The path is said to be unidirectional, i.e. one room is only connected to another room. For instance, room 1 is connected to room 3, but room 3 is not connected to room 1. In the other hand, there are rooms that are connected to themselves, so they don't have an exit.

For instance:

- 0 → 0 : has an exit by definition
- 1 → 3 : has an exit via 3
- 2 → 1 : has an exit via 1, then 3
- 3 → 0 : has an exit directly connected to 0
- 4 → 2 : has an exit via 2, then 1, then 3
- 5 → 5 : do not has an exit

Santa would like to know how many rooms, that are connected to the exit, are so he can continue his journey. In the previous example, we would give Santa the answer of 5 (5 rooms with an exit). All of them but room 5.

Would you help Santa to find the way out so he can deliver all the presents?

### Input

<n> positive integer number greater than 0 indicating how many rooms there are.

<room ID> <room ID connection> for every room we give the room id and its connection. The room ID will be between 0 and n-1. The given sequence can be in any order.

### Output

Print out the positive integer indicating how many rooms have an exit.

## Example

### Input

```
6
0 0
1 3
2 1
3 0
4 2
5 5
```

### Output

```
5
```



## Solutions

### Python3

```
EXIT_ROOM = 0 # exit room is marked with a 0

# solve the problem
def solve(rooms):
    sol = 1

    for i in range(1, len(rooms)):
        visited = [0] * len(rooms)
        visited[0] = 1

        j = i
        # walk through the rooms
        # and test if we can find the exit
        while not visited[j]:
            visited[j] = 1

            if rooms[j] == EXIT_ROOM:
                sol += 1

            j = rooms[j]

    return sol

# read the total number of rooms
n = input()
n = int(n)

# initialize the rooms
rooms = [-1] * n

# read the connection between rooms
for i in range(n):
    line = input()
    line = line.split()

    rooms[int(line[0])] = int(line[1])

sol = solve(rooms)
print(sol)
```

## C++

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

const int EXIT_ROOM = 0; // exit room is marked with a 0

int solve(vector<int>& rooms) {
    int j, sol = 1;
    for (int i = 1; i < rooms.size(); i++) {
        vector<int> visited(rooms.size(), 0);
        visited[0] = 1;
        j = i;
        while (!visited[j]) {
            visited[j] = 1;

            if (rooms[j] == EXIT_ROOM) {
                sol += 1;
            }

            j = rooms[j];
        }
    }
    return sol;
}

int main() {
    int n, i, id1, id2, sol;
    // read total number of rooms
    cin >> n;
    // initialize the rooms
    vector<int> rooms(n, -1);
    // read the connections between rooms
    i = 0;
    while (i++ < n) {
        cin >> id1 >> id2;
        rooms[id1] = id2;
    }
    sol = solve(rooms);
    cout << sol << endl;
}
```

## 26 Skiing

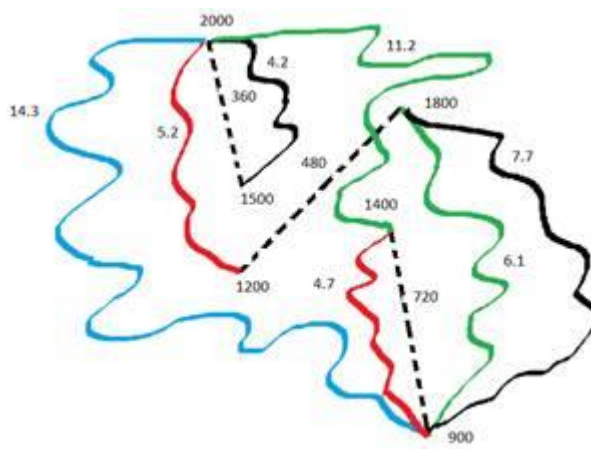
20 points

### Introduction

A ski station consists of lifters and slopes.

Lifters bring you uphill from a lower starting point to a higher ending point, while slopes are the other way around. Lifters should have at least one slope connected to its starting and ending point.

Slopes have a difficulty level, which limits the maximum speed a skier can reach on them: 20 km/h on green slopes, 30 km/h on blue slopes, 40 km/h on red slopes, and 50 km/h on black slopes. Slopes can start or end in another slope or in a lifter. Multiple slopes and lifters can be connected.



Given a ski domain map, which contains lifters and slopes, determine the minimum time (in seconds) needed to go from the top of the ski station to the bottom.

Note that, for the sake of simplicity, we cannot have two or more starting and ending points with the same height. Moreover, in case of trouble, we can always call the relief helicopter, which will bring us to the bottom of the ski resort in two hours.

### Input

The map should be read as follows: The first number in the input is the number of lifters and slopes the ski station has. Then, for each element, the input provides a letter that specifies if it is a lifter (L) or a slope (S). For lifters, following we have the time a lifter requires to reach its end in seconds, followed by its starting point and ending point height, both given in meters. For slopes, we have the difficulty level, the length in kilometers, and the starting point and the ending point heights, both given in meters as well.

### Output

Print out the minimum time in seconds with a resolution of two decimals needed to go from the top of the ski station to the bottom.

## Example

### Input

```
10
L 360 1500 2000
L 480 1200 1800
L 720 900 1400
S black 4.2 2000 1500
S red 5.2 2000 1200
S blue 14.3 2000 900
S green 11.2 2000 1400
S red 4.7 1400 900
S black 7.7 1800 900
S green 6.1 1800 900
```

### Output

```
1502.40
```

**Note: in this example the fastest way down is 2000 – red slope – 1200 – lifter – 1800 – black slope – 900**



## Solutions

## Python3

```
import sys
from queue import PriorityQueue

slope_speed = { "black": 50.0, "red": 40.0,
                 "blue": 30.0, "green": 20.0}

def slopeTime(color, length):
    return float(length / slope_speed[color] * 3600.0)

def readInput(g):
    n = int(input())
    last_id = -1
    #we will calculate the start and end po for the
    #algorithm while reading the input
    maxHeight = 0
    minHeight = sys.maxsize
    for _ in range(0, n):
        time = 0
        line = input().split()
        type = line[0]
        start = -1
        end = -1
        if(type == "L"):
            time = int(line[1])
            start = int(line[2])
            end = int(line[3])
        else:
            color = line[1]
            length = float(line[2])
            time = slopeTime(color, length)
            start = int(line[3])
            end = int(line[4])

        #update min and max heights
        maxHeight = max(start, end, maxHeight)
        minHeight = min(start, end, minHeight)

        #add the time from start to end to the g
        if not start in g:
            g[start] = {}
        if not end in g:
            g[end] = {}
        if not end in g[start]:
            g[start][end] = time
        else:
            #if there is more than one way from start
            #to end we store the fastest one
            g[start][end] = min(time, g[start][end])
    t = (maxHeight, minHeight)
    return t
#continues on next page...
```



```
#...
def dijkstra(g, start, end):
    # You can go anywhere in at least 2 hours (helicopter)
    max_time = 2*3600
    n = len(g)
    # Store the minimum time to go from the start to any
    # other node (in the worst case helicopter)
    min_times = {}
    for v in g:
        min_times[v] = max_time
    #the time from the start to the start is always 0
    min_times[start] = 0
    pq = PriorityQueue()
    #pre-populate pq
    for v in min_times:
        pq.put((min_times[v], v))

    while not pq.empty():
        v = pq.get()
        #v_id is on [1] because of the pq implementation
        v_id = v[1]
        #for each neighbor u of v
        for u_id in g[v_id]:
            time = g[v_id][u_id]
            # if the minimum time we have found until now to go from
            # start to u is greater than the time from start to u
            # passing through v we update the minimum time to this
            # value and push u the queue with updated time
            if(min_times[u_id] > min_times[v_id]+time):
                min_times[u_id] = min_times[v_id]+time
                pq.put((min_times[u_id], u_id))
        # return the min time from start to end
    return min_times[end]

g = {}
# read the input and construct the graph
maxmin = readInput(g)
start = maxmin[0]
end = maxmin[1]
# Compute time using dijkstra's algorithm
time = dijkstra(g, start , end)
print( "%.2f" % time)
```

## C++

```
#include <iostream>
#include <string>
#include <iomanip>
#include <map>
#include <vector>
#include <queue>
using namespace std;
#define MAX_INT 2147483647;

typedef pair<int, float> Node;
typedef vector< Node > Row;
typedef vector<Row> Graph;
typedef map<int, int> Dict;

//Comp class to compare pairs in the priority queue
struct Comp {
    bool operator () (const Node &a,
                     const Node &b){
        if (a.second == b.second)
            return a.first < b.first;
        return a.second < b.second;
    }
};

map<string, float> slope_speed{
    {"black",50}, {"red",40}, {"blue",30}, {"green",20}
};

float slopeTime(const string& colour, float length){
    return length / slope_speed[colour] * 3600.0;
}

void addVertex(int h, int id, Graph& g, Dict& h2id){
    //generate a new ID for vertex at from height
    //and increment the id counter
    h2id[h] = id;
    Row empty_row = Row();
    //add the vector position for this id
    g.push_back(empty_row);
}
//continues on next page...
```

```
//...
pair<int, int> readInput(Graph& g, Dict& h2id){
    int n, last_id = -1;
    cin >> n;
    //we will calculate the start and end points for the
    //algorithm while reading the input
    int maxHeight = 0, minHeight = MAX_INT;
    for(int i = 0; i < n; i++){
        char type;
        float time;
        cin >> type;
        if(type == 'L'){
            cin >> time;
        }
        else{
            string color;
            float length;
            cin >> color >> length;
            time = slopeTime(color, length);
        }
        int start, end;
        cin >> start >> end;
        //update min and max heights
        maxHeight = max(start, max(end, maxHeight));
        minHeight = min(start, min(end, minHeight));
        // If the start height is a new height
        if(h2id.find(start) == h2id.end()){
            addVertex(start, ++last_id, g, h2id);
        }
        // If the end height is a new height
        if(h2id.find(end) == h2id.end()){
            addVertex(end, ++last_id, g, h2id);
        }
        //add the time from start to end to the g
        int start_id = h2id[start], end_id = h2id[end];
        g[start_id].push_back(make_pair(end_id, time));
    }
    return make_pair(maxHeight, minHeight);
}
//continues on next page...
```

```
//...
float dijkstra(const Graph& g, int start, int end){
    // You can go anywhere in at least 2 hours (helicopter)
    float max_time = 2 * 3600;
    int n = g.size();
    // The (default) minimum time to go from
    // the start to any other node (in the worst case)
    vector<float> min_times(n, max_time);
    //the time from the start to the start is always 0
    min_times[start] = 0;
    priority_queue<Node, Row, Comp> pq;
    //pre-populate pq
    for(int i = 0; i < min_times.size(); ++i){
        pq.push(make_pair(i, min_times[i]));
    }
    while(!pq.empty()){
        Node v = pq.top();
        int v_id = v.first;
        pq.pop();
        //for each neighbor u of v
        for(int i = 0; i < g[v_id].size(); ++i){
            Node u = g[v_id][i];
            int u_id = u.first;
            float time = u.second;
            // if the minimum time we have found until now to go from
            // start to u is greater than the time from start to u
            // passing through v we update the minimum time to this
            // value and push u into the queue with updated time
            if(min_times[u_id] > min_times[v_id]+time){
                min_times[u_id] = min_times[v_id]+time;
                pq.push(make_pair(u_id, min_times[u_id]));
            }
        }
    }
    return min_times[end];
}

int main(){
    Graph g;
    Dict h2id;
    // read the input and construct the graph
    pair<int, int> maxmin = readInput(g, h2id);
    int start = h2id[maxmin.first];
    int end = h2id[maxmin.second];
    float time = dijkstra(g, start , end) ;
    cout << setprecision(2) << fixed << time << endl;
}
```

## 27 Digital Castellers

20 points

### Introduction

A *Castell* is a human tower built traditionally in festivals at many locations within Catalonia. People that forms part of a *Castell* are known as *castellers*.

So, in our a computer programming contest you are requested to build digital castellers.

There are three definite parts of a digital castle; the *pinya*, the *tronc*, and the *pom de dalt* or the crown of the castle.

The *pinya* is the base of the *Castell*, and it's composed by:

- The *soca*, in the first floor
- The *folre*, in the second floor. It's optional.
- The *manilles*, in the third floor. It's optional, and only allowed in top of a *folre*.

The *tronc* is the main visible structure of the *Castell*, and it's between the *pinya* and the *pom de dalt*. The number of *castellers* in each floor of the *tronc* is defined in the name of the *Castell*.

The *pom de dalt* is the top of the *Castell* and it's composed by:

- The *dosos*, two *castellers* just on top of the *tronc*.
- The *acotxador*, one *casteller* on top of the *dosos*.
- The *enxaneta*, who crowns the *Castell*, on top of the *acotxador*.

In addition, a *Castell* may have an *agulla*. This is a central tower in the centre of the *Castell*, with the same height as the *tronc*.

The most common nomenclature to describe a *Castell* is, in Catalan

$N$  de  $M$  [amb  $X$ ] [i  $Y$ ] [i  $Z$ ]

Where  $N$ ,  $M$ ,  $X$ ,  $Y$  and  $Z$  are the variables that define the structure of the *Castell*.

- $N$  is the number of *castellers* in each floor of the *tronc*, with  $1 \leq N \leq 10$
- $M$  is the height of the *Castell* in *castellers*, with  $3 \leq M \leq 10$
- $X$ ,  $Y$  and  $Z$  are optional, and they may be, in that order
  - *folre*
  - *manilles*
  - *l'agulla*



N and M are expressed in catalan with the following nomenclature

- Only for N
  - 1 = *pilar*
  - 2 = *dos* or *torre*
  
- For N and M
  - 3 = tres
  - 4 = quatre
  - 5 = cinc
  - 6 = sis
  - 7 = set
  - 8 = vuit
  - 9 = nou
  - 10 = deu

The Castells constructed by a Pilar (N = 1) does not have the *dosos* nor the *acotxador*.

The problem consists on, given a Castell description, represent it graphically

- Each *casteller* is represented with a # character
- The *Castell* has to be as symmetrical as possible
- If the total width of the *tronc* is odd, there will be a space between the *dosos*.
- If the total width of the *tronc* is even, the *acotxador* and *l'enxaneta* will be in the left.

### Input

A string with the definition of the castell to draw.

### Output

The digital castell

#### Example 1

#### Input

quatre de set amb folre

#### Output

```
#
#
##
####
####
#####
#####
```

## Example 2

### Input

tres de sis

### Output

```
#
#
# #
###
###
#####
```

## Example 3

### Input

cinc de nou amb folre i manilles

### Output

```
#
#
# #
#####
#####
#####
#####
#####
#####
#####
```



## Solutions

## Python3

```
import math

wordToNumber = {
    "pilar"      : 1, "torre"    : 2, "dos"      : 2,
    "tres"       : 3, "quatre"  : 4, "cinc"     : 5,
    "sis"        : 6, "set"     : 7, "vuit"     : 8,
    "nou"        : 9, "deu"     : 10
}

# Precondition: the command is correctly constructed
def parseCastell(nomCastell):
    words = nomCastell.lower().split()
    # Get n (people per floor)
    n = wordToNumber[words[0]]
    # Skip words[1] = de
    # Get m (number of floors)
    m = wordToNumber[words[2]]
    folre = "folre" in words
    manilles = "manilles" in words
    agulla = "l'agulla" in words
    return n, m, folre, manilles, agulla

def buildCastell (n, m, ambFolre, ambManilles, ambAgulla):
    # width "pinyes"
    soca = n + 2 + ambAgulla
    folre = 0
    manilles = 0

    if (ambFolre):
        soca = n + 4 + ambAgulla
        folre = n + 2 + ambAgulla

    if (ambManilles):
        soca = n + 6 + ambAgulla
        folre = n + 4 + ambAgulla
        manilles = n + 2 + ambAgulla

    castell = "#" * soca

    offset = 0
    if (ambFolre):
        offset = offset + 1
        castell = " " * offset + "#" * folre + "\n" + castell
    # continues on next page ...
```



```
#...
if (ambManilles):
    offset = offset + 1
    castell = " " * offset + "#" * manilles + \
        "\n"+ castell

    offset = offset + 1

# Tronc
if (n > 1):
    tronc = m - 3 - ambManilles - ambFolre - 1

    for i in range(tronc):
        castell = " " * offset + "#" * (n+ambAgulla) + \
            "\n"+ castell
    # Dossos
    offset = math.floor(offset + ((n+ambAgulla) / 2)) - 1
    if (n+ambAgulla)%2 == 0:
        castell = " " * offset + "#" * 2 + "\n"+ castell
    else:
        castell = " " * offset + "# #" + "\n"+ castell
        offset = offset + 1
    #Acotxador
    castell = " " * offset + "#" + "\n"+ castell
    #Enxaneta
    castell = " " * offset + "#" + "\n"+ castell

else:
    tronc = m - ambManilles - ambFolre - 1
    for i in range(tronc):
        castell = " " * offset + "#" * (n+ambAgulla) + \
            "\n"+ castell

    return castell

def main():
    nomCastell = input()
    n, m, folre, manilla, agulla = parseCastell(nomCastell)
    castell = buildCastell(n, m, folre, manilla, agulla)
    print(castell)

main()
```

## C++

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

map<string, int> catalan_number = {
    {"pilar", 1}, {"dos", 2}, {"torre", 2},
    {"tres", 3}, {"quatre", 4}, {"cinc", 5},
    {"sis", 6}, {"set", 7}, {"vuit", 8},
    {"nou", 9}, {"deu", 10}
};

void parseInput(int& n, int& m,
               bool& folre, bool& manilles, bool& agulla){
    string N, de, M, token;
    cin >> N >> de >> M; //N de M
    if(cin >> token){ //amb
        cin >> token; //folre
        folre = true;
        if(cin >> token){//i
            cin >> token; //manilles
            if(token == "manilles"){
                manilles = true;
                if(cin >> token){//i
                    cin >> token; //l'agulla
                    agulla = true;
                }
            }
        }
        else{
            agulla = true;
        }
    }
    n = catalan_number[N]; m = catalan_number[M];
}

void drawCastell(int pinya, int troncH, int troncW){
    int m = pinya+troncH+3;
    if(troncW == 1) m = pinya+troncH+1;
    vector<string> castell = vector<string>(m);
    //pinya
    for(int i = pinya-1; i >= 0; --i){
        string nivell = "";
        //spaces
        for(int j = 0; j < i; ++j){
            nivell += ' ';
        }
        //castellers
        for(int j = 0; j < troncW+2*pinya-2*i; ++j){
            nivell+='#';
        }
        castell[i] = nivell;
    }
    //continues on next page...
```

```
//...

//tronc
for(int i = pinya; i < pinya+troncH; ++i){
    string nivell = ""; //spaces
    for(int j = 0; j < pinya; ++j){
        nivell += ' ';
    } //catsellers
    for(int j = 0; j < troncW; ++j){
        nivell+='#';
    }
    castell[i] = nivell;
}
//dossos are not present on pilars
if(troncW > 1){
    string dossos; //spaces
    for(int j = 0; j < pinya+(troncW/2)-1; ++j){
        dossos += ' ';
    } //castellers
    if(troncW % 2 != 0)
        dossos += "# #";
    else dossos += "##";
    castell[pinya+troncH] = dossos;
}
//acotxador and enxaneta
string acotxaneta;
int num_espais = pinya+(troncW/2);
if(troncW % 2 == 0) --num_espais;
//spaces
for(int j = 0; j < num_espais; ++j)
    acotxaneta += ' ';
//castellers
acotxaneta += "##";
//acotxadors are not present in pilars
if(troncW > 1){
    castell[pinya+troncH+1] = acotxaneta;
    castell[pinya+troncH+2] = acotxaneta;
}
else
    castell[pinya+troncH] = acotxaneta;
//print castell
for(int i = m-1; i >= 0; --i)
    cout << castell[i] << endl;
}

int main(){
    int n,m;
    bool folre = false, manilles = false, agulla = false;
    parseInput(n,m,folre,manilles,agulla);
    int pinyaH = 1 + int(folre) + int(manilles);
    int troncW = n + int(agulla);
    int troncH = m - pinyaH - 3;
    if(troncW == 1) troncH = m - pinyaH - 1;
    drawCastell(pinyaH,troncH,troncW);
}
```

## 28 Meowy's Island

20 points

### Introduction

Meowy, the adventurer cat, has found a new island to live during his journeys. However, this island has a very irregular terrain and it is quite singular: the sea level grows rapidly during the day. The problem is that Meowy hates water, although he thinks that the island is really beautiful. So he needs to know the level of the sea at every hour in order to avoid flooded areas.

Meowy has studied the island carefully, thus he knows the following:

- There is a map of the island represented as a grid, where the sea cells are marked with '.' and the island cells contain a height value in the interval [1,9].
- When the sea level increases, the water from a cell floods the neighboring cells (consider the 8 directions: north, north-east, east, south-east, south, south-west, west, north-west) only if the height of the neighboring cell is less or equal than the sea level.
- There might be holes in the island (cells with lesser height than the surrounding cells, see top right part of the island in the Example 2). Holes are flooded only when at least one of the surrounding higher cells are covered by the sea.
- There might be sea cells inside the island (see bottom left part of the island in Example 2).

### Input

The input of the program is:

- < The rows of the map. >
- < The columns of the map. >
- < The values of the height map. >
- < The initial sea level. >
- < The final sea level. >

### Output

The state of the map at every height level of the sea. Cells with water are represented by a 'W' and cells without water are represented with a space ' '.















## Solutions

## Python3

```
def getNeighbours(cell, nRows, nCols):
    # Returns the list of neighbours of a cell
    (row, col) = cell
    listNeighbours = []
    for i in [-1, 0, 1]:
        for j in [-1, 0, 1]:
            if 0 <= i + row < nRows and 0 <= j + col < nCols:
                listNeighbours.append((row + i, col + j))
    return listNeighbours

def expand(cell, heights, final, visited, level):
    # From cell, it expands the water to the other cells.
    # Cell has always water.
    if cell in visited: return
    nRows = len(heights)
    nCols = len(heights[0])
    visited.add(cell)
    listNeighbours = getNeighbours(cell, nRows, nCols)
    for (i, j) in listNeighbours:
        if heights[i][j]== '.' or int(heights[i][j]) <= level:
            final[i][j] = 'W'
            expand((i, j), heights, final, visited, level)

def calculateMap(heights, level):
    # calculates the resulting map from a heights
    # and a level of water
    nRows = len(heights)
    nCols = len(heights[0])
    visited = set()
    final = [[' ']* nCols for _ in range(nRows)]
    for i in range(nRows):
        for j in range(nCols):
            if heights[i][j] == '.':
                expand((i, j), heights, final, visited, level)
    return final

# continues on next page...
```

```
# ...
def checkAnswer(heights, final, level):
    # Checks if a map representing a solution of the
    # problem (final) is OK
    nRows = len(heights)
    nCols = len(heights[0])
    for i in range(nRows):
        for j in range(nCols):
            if final[i][j] == 'W':
                meAndNeighbours = getNeighbours((i, j),\
                                                nRows, nCols) + [(i, j)]
                assert any((heights[a][b] == '.'\
                            or int(heights[a][b]) <= level\
                            for (a, b) in meAndNeighbours))
            else:
                if int(heights[i][j]) <= level:
                    neighbours = getNeighbours((i,j), nRows, nCols)
                    assert all((heights[a][b] != '.'\
                                for (a, b) in neighbours))

# get input data
nRows = int(input())
nCols = int(input())
heights = []
for _ in range(nRows):
    heights.append(input())
iniLevel = int(input())
endLevel = int(input())

for level in range(iniLevel, endLevel + 1):
    final = calculateMap(heights, level)
    # print map
    print("Sea level:", level)
    for row in final:
        print(''.join(row))
```

## C++

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

typedef vector<bool> RowBool;
typedef vector<RowBool> MatrixBool;

typedef vector<int> RowInt;
typedef vector<RowInt> MatrixInt;

struct Position{
    int i; int j;
    Position(int ii, int jj) : i(ii), j(jj) {}
};

void printMap(const MatrixBool& map, int level)
{
    cout << "Sea level: " << level << endl;
    for (int i = 0; i < map.size(); ++i){
        for (int j = 0; j < map[i].size(); ++j){
            cout << (map[i][j]? 'W': ' ');
        }
        cout << endl;
    }
}

void applyFlood(const MatrixInt & map, int level, MatrixBool& flooded,
int row, int col){
    stack<Position> st;
    st.push(Position(row, col));
    while (!st.empty()){
        Position pos = st.top(); st.pop();
        flooded[pos.i][pos.j] = true;
        // Flood previous row
        if (pos.i > 0){
            if (pos.j > 0 && !flooded[pos.i-1][pos.j-1] &&
                map[pos.i-1][pos.j-1] <= level){
                st.push(Position(pos.i-1, pos.j-1));
            }
            if (!flooded[pos.i-1][pos.j]
                && map[pos.i-1][pos.j] <= level){
                st.push(Position(pos.i-1, pos.j));
            }
            if (pos.j+1 < map[0].size() && !flooded[pos.i-1][pos.j+1]
                && map[pos.i-1][pos.j+1] <= level){
                st.push(Position(pos.i-1, pos.j+1));
            }
        }

        // Flood current row
        if (pos.j > 0 && !flooded[pos.i][pos.j-1]
            && map[pos.i][pos.j-1] <= level){
            st.push(Position(pos.i, pos.j-1));
        }
    }
    //continues on next page...
}
```

```
//...
    if (pos.j+1 < map[0].size() && !flooded[pos.i][pos.j+1]
        && map[pos.i][pos.j+1] <= level){
        st.push(Position(pos.i, pos.j+1));
    }

    // Flood next row
    if (pos.i+1 < map.size()){
        if (pos.j > 0 && !flooded[pos.i+1][pos.j-1]
            && map[pos.i+1][pos.j-1] <= level){
            st.push(Position(pos.i+1, pos.j-1));
        }
        if (!flooded[pos.i+1][pos.j]
            && map[pos.i+1][pos.j] <= level){
            st.push(Position(pos.i+1, pos.j));
        }
        if (pos.j+1 < map[0].size()
            && !flooded[pos.i+1][pos.j+1]
            && map[pos.i+1][pos.j+1] <= level){
            st.push(Position(pos.i+1, pos.j+1));
        }
    }
}

MatrixBool applySeaLevel(const MatrixInt& map, int level){
    MatrixBool flooded(map.size(),
                       RowBool(map[0].size(), false));
    // Loop over the map, start a flood from every "sea" cell
    for (int i = 0; i < map.size(); ++i)
        for (int j = 0; j < map[i].size(); ++j)
            if (!flooded[i][j] && map[i][j] == 0)
                applyFlood(map, level, flooded, i, j);
    return flooded;
}

int main(){
    int rows = 0, cols = 0;
    cin >> rows >> cols;
    MatrixInt map(rows, RowInt(cols, 0));
    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < cols; ++j){
            char c;
            cin >> c;
            if (c >= '1' && c <= '9')
                map[i][j] = int(c - '0');
        }
    int minSeaLevel, maxSeaLevel;
    cin >> minSeaLevel >> maxSeaLevel;
    for (int level = minSeaLevel; level <= maxSeaLevel; ++level)
        printMap(applySeaLevel(map, level), level);
}
```

## 29 Hexagons

23 points

### Introduction

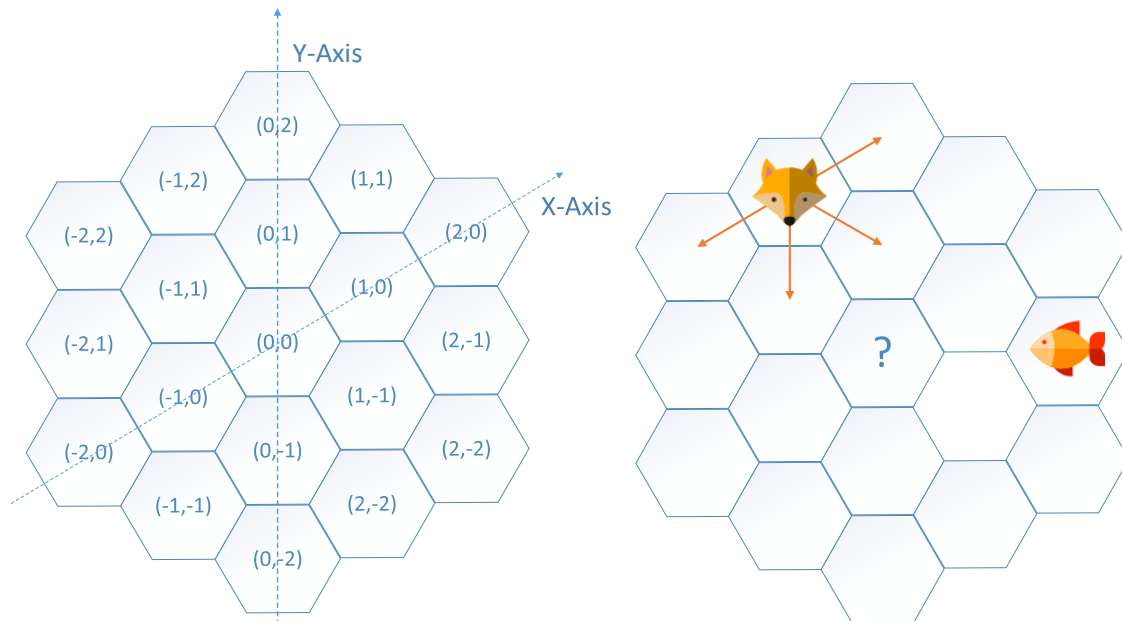
Clusters of hexagonal columns known as the Giant's Causeway can be found along the coast of Northern Ireland. Legends claim the causeway was built by Irish giant Finn MacCool, who had been challenged to a fight by Scottish giant Benandonner.



The columns vary in height, and some have a small hole in the top. When the waves break with the columns, sometimes fishes get trapped in the holes.

Foxes live in the area, and they love to eat fish. To get the trapped fishes, a fox has to move above the columns. Each fox has two characteristics: the maximum jump, and the maximum fall it can survive. A fox only can jump to a near hexagon if its differential height is less or equal to its maximum jump, and only can fall to a near hexagon if the differential height is less or equal to its maximum fall it can survive.

We can identify each hexagonal column using a  $(x,y)$  notation using the following convention:



We want to know if a fox positioned in a certain hexagon is able to get a certain fish.

### Input

- Multiple lines, each one with a hexagon x,y coordinates and its height
- Line with end character #
- Multiple lines, each one with the maximum jump, the maximum fall and the fox and the fish coordinates

### Output

For each line with a fox and fish description:

- If exists a path: "The fox says: what a delicious fish!"
- If not: "The fish says: not today, little fox!"



## Example

### Input

```
0 0 0
0 1 1
0 2 2
0 3 5
0 4 0
#
0 0 0 0 0 0
1 1 0 0 0 2
1 1 0 0 0 3
3 1 0 0 0 3
3 1 0 0 0 4
3 5 0 0 0 4
```

### Output

```
The fox says: what a delicious fish!
The fox says: what a delicious fish!
The fish says: not today, little fox!
The fox says: what a delicious fish!
The fish says: not today, little fox!
The fox says: what a delicious fish!
```



## Solutions

## Python3

```
class HexagonBoard:
    def __init__(self):
        self.hexagons = {}

    def addHexagon(self, x, y, height):
        self.hexagons[x,y] = height

    def getNeighbors(self, x, y):
        return [(x,y+1), (x+1,y), (x+1,y-1), (x,y-1), (x-1,y), (x-1,y+1)]

    def existsPath(self, xs, ys, xt, yt, maxJump, maxFall):
        # set of accesible hexagons but not explored yet
        explored = set()
        toExplore = {(xs,ys)}
        # True if a path exists betwen (xs,ys) and (xt,yt)
        found = False
        while (not found) and (len(toExplore) > 0):
            # Get an unexplored hexagon
            (cx,cy) = toExplore.pop()
            # Check if it's the target
            if (cx,cy) == (xt, yt):
                found = True
            else:
                # Add the accesible hexagons to the list toExplore
                for i,j in self.getNeighbors(cx,cy):
                    # Check if neighbours hexagons has been defined
                    # and has not been explored
                    if (i,j) in self.hexagons and not (i,j) in explored:
                        height = self.hexagons[i,j] - self.hexagons[cx,cy]
                        if height <= maxJump and height >= maxFall:
                            toExplore.add((i,j))
                    # Mark the hexagon as explored (not explore it again)
                    explored.add((cx,cy))
        return found
# continues on next page...
```

```
# ...

def main():
    hb = HexagonBoard()
    # Read hexagons and heights
    line = input()
    while line != "#":
        nums = line.split()
        x = int(nums[0])
        y = int(nums[1])
        height = int(nums[2])
        hb.addHexagon(x,y,height)
        line = input()

    # Read paths to find until end of file
    line = input()
    while True:
        nums = line.split()
        maxJump = int(nums[0])
        maxFall = -int(nums[1])
        xs = int(nums[2])
        ys = int(nums[3])
        xt = int(nums[4])
        yt = int(nums[5])

        if hb.existsPath(xs,ys,xt,yt, maxJump, maxFall):
            print("The fox says: what a delicious fish!")
        else:
            print("The fish says: not today, little fox!")
        try:
            line = input()
        except EOFError:
            break

main()
```

## C++

```
#include <iostream>
#include <vector>
#include <set>
#include <sstream>
#include <map>
#include <string>
using namespace std;

class Hexagon{
public:
    int x, y, h;
    set<Hexagon*> links;
    Hexagon(int x, int y, int h){
        this->x = x;
        this->y = y;
        this->h = h;
    }
};

bool isWayFromFoxToFish(Hexagon* hex, set<Hexagon*> &path,
    int jump, int fall, int fix, int fiy){
    bool reach = false;
    if (path.count(hex) > 0) return false;
    if (hex->x == fix && hex->y == fiy) return true;

    path.insert(hex);

    set<Hexagon*>::iterator it;
    for (it = hex->links.begin();
        it != hex->links.end() && !reach; ++it){

        if (*it != hex){
            int gap = (*it)->h - hex->h;
            if ((gap == 0) || (gap > 0 && gap <= jump)
                or (gap < 0 && -gap <= fall))
                reach |= isWayFromFoxToFish(*it, path, jump,
                    fall, fix, fiy);
        }
    }
    path.erase(hex);
    return reach;
}

// continues on next page...
```

```
// ...
void insertAndConnect(map<string, Hexagon*> &grid, int x,
                    int y, int h){
// Insert
    Hexagon *hex = new Hexagon(x, y, h);
// Connect
    for ( int i = -1; i <= 1; i++ ){
        for (int j = -1; j <= 1; j++){
            if ((i+j > -2) && (i+j < 2) && ((i+x) != x
                || (j+y) != y)){
                ostreamstream s;
                s << x+i << "-" << y+j;
                string key(s.str());
                map<string, Hexagon*>::iterator it;
                it = grid.find(key);
                if (it != grid.end()){
                    it->second->links.insert(hex);
                    hex->links.insert(it->second);
                }
            }
        }
    }
    ostreamstream s;
    s << x << "-" << y;
    string key(s.str());
    grid[key] = hex;
}

int main(){
    map<string, Hexagon*> grid;

    string x;
    int y, h;
    cin >> x;
    while (x != "#"){
        cin >> y >> h;
        insertAndConnect( grid, stoi(x), y, h);
        cin >> x;
    }

    set<Hexagon*> path;
    int maxjump, maxfall, foxx, foxy, fishx, fishy;
    while (cin >> maxjump >> maxfall >> foxx >> foxy
           >> fishx >> fishy){
        ostreamstream s;
        s << foxx << "-" << foxy;
        string key(s.str());
        path.clear();

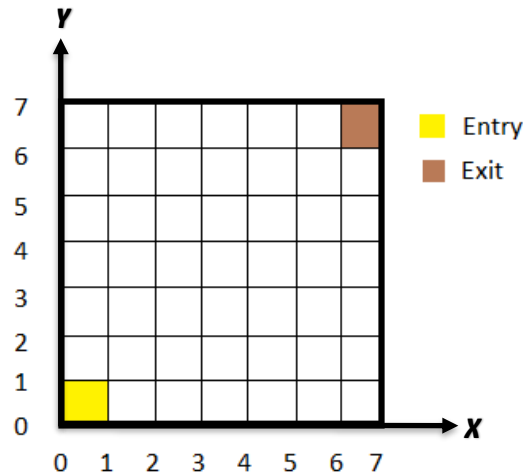
        if (isWayFromFoxToFish(grid[key], path, maxjump,
                                maxfall, fishx, fishy))
            cout << "The fox says: what a delicious fish!";
        else cout << "The fish says: not today, little fox!";
        cout << endl;
    }
}
```

# 30 Maze

23 points

## Introduction

Calculate the shortest path from the entry square to the exit square in a given maze of size  $DimX \times DimY$ . The entry will be in the bottom left of the maze and the exit in the upper right of maze. The movements along the maze can be one square up, down, right or left.



In this example it is shown a maze of  $7 \times 7$



### RESTRICTIONS:

$DimX > 1$

$DimY > 1$

$N \geq 0$

$0 \leq OX_n, DX_n \leq DimX$

$0 \leq OY_n, DY_n \leq DimY$

## Input

For each test, the first line will be  $DimX$ , number of columns of the maze.

The second line will be  $DimY$ , number of rows of the maze.

The third line will be  $N$ , number of walls in the maze.

The next  $N$  lines will be " $OX_n, OY_n, DX_n, DY_n$ ", the coordinates of the origin and the destiny of the wall  $n$ .

## Output

The output must one line. It will contain the distance from the entry to the exit. In case it is impossible to reach the exit the output will be -1.

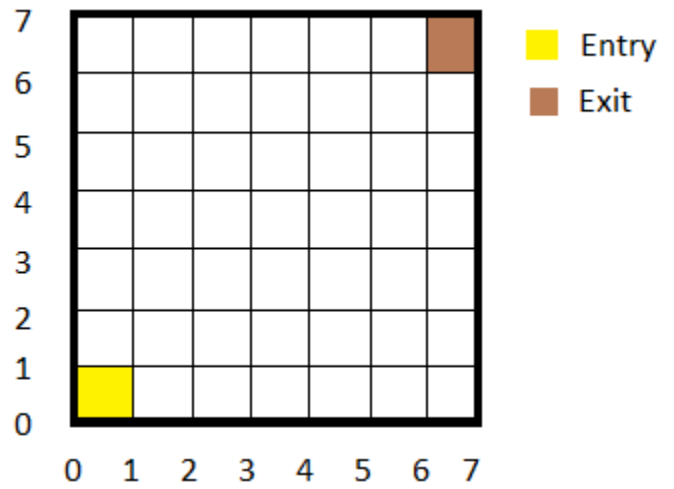
### Example 1

**Input**

7  
7  
7  
0

**Output**

12



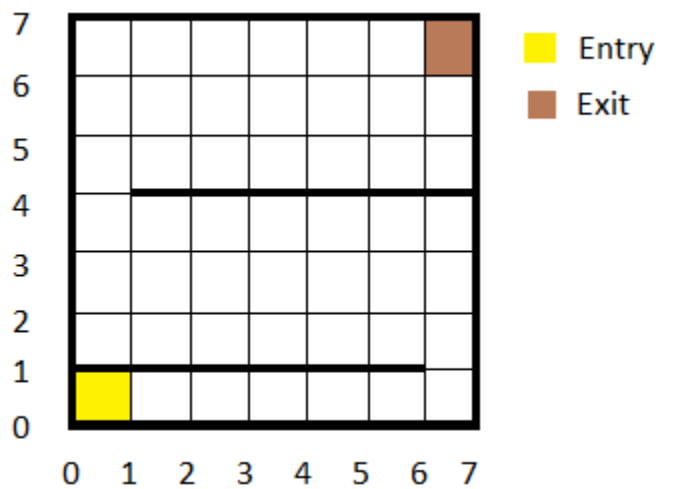
### Example 2

**Input**

7  
7  
2  
0,1,6,1  
1,4,7,4

**Output**

24



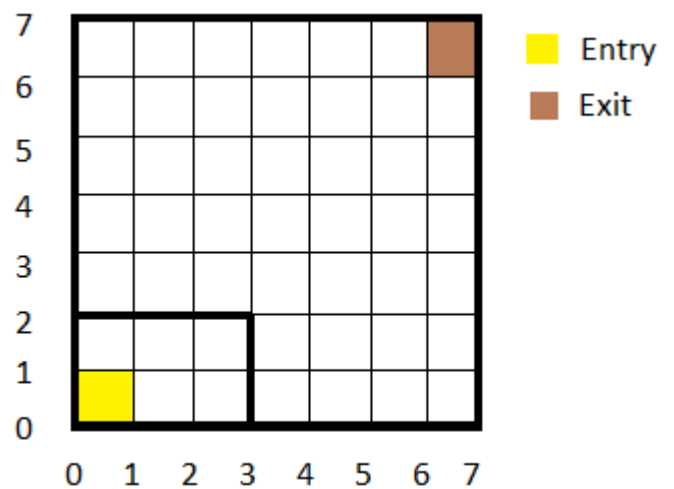
### Example 3

**Input**

7  
7  
2  
0,2,3,2  
3,2,3,0

**Output**

-1



## Solutions

## Python3

```
import queue

dirs = [
    {'x': 1, 'y': 0}, {'x': -1, 'y': 0},
    {'x': 0, 'y': 1}, {'x': 0, 'y': -1}
]

class Cell(object):
    def __init__(self, x, y, end = False):
        self.x = x
        self.y = y
        self.visited = False
        self.distance = 0
        self.end = end

    def visit(self, distance):
        self.visited = True
        self.distance = distance + 1

    def wall(self, line, eje, posEje, posPerp, positive):
        perp = not eje
        return line[0+eje] == posEje+positive\
            and line[2+eje] == posEje+positive\
            and ((line[0+perp] <= posPerp\
                and line[2+perp] >= (posPerp+1))\
                or (line[2+perp] <= posPerp\
                and line[0+perp] >= (posPerp+1)))

    def move(self, dir, bloqueos, dx, dy):
        mx = dir['x']; my = dir['y']
        moves = [{'along' : self.x, 'perp' : self.y,\
                  'dir' : mx, 'dist' : dx},\
                {'along' : self.y, 'perp' : self.x,\
                  'dir' : my, 'dist' : dy}]
        for mov in moves:
            if mov['dir'] != 0:
                if mov['along']+mov['dir'] <= mov['dist'] - 1\
                    and mov['along']+mov['dir'] >= 0:
                    for line in bloqueos:
                        if self.wall(line, abs(my), mov['along'],\
                                    mov['perp'], max(mov['dir'], 0)):
                            return False
                    return True
            return False

dx = int(input()); dy = int(input())
```

```
num_lines = int(input())
lines = []

for i in range(num_lines):
    inp = input()
    inp_array = inp.split(',')
    lines.append((int(inp_array[0]), int(inp_array[1]), \
                  int(inp_array[2]), int(inp_array[3])))

maze=[]
for x in range(dx):
    column = []
    for y in range(dy):
        cell = Cell(x,y)
        column.append(cell)
    maze.append(column)

# Last cell
maze[dx-1][dy-1] = Cell(dx-1, dy-1, True)

my_q = queue.Queue()
# First cell
my_q.put(maze[0][0])
distance = -1
maze[0][0].visit(distance)
end = False

while not my_q.empty() and not end:
    cell = my_q.get()
    distance = maze[cell.x][cell.y].distance
    end = end or maze[cell.x][cell.y].end

    for dir in dirs:
        canMove = cell.move(dir, lines, dx, dy)
        if canMove:
            mx = dir['x']; my = dir['y']
            if (not maze[cell.x+mx][cell.y+my].visited):
                my_q.put(maze[cell.x+mx][cell.y+my])
                maze[cell.x+mx][cell.y+my].visit(distance)

if not end:
    distance = -1
print(distance)
```



## C++

```
#include <algorithm> //minmax
#include <iostream>
#include <queue>
#include <tuple> //tie
#include <vector>
using namespace std;

struct Cell{
    int d = 0; //the distance from the start to the cell
    int x; int y; //the coordinates of the cell
};

typedef vector<Cell> Elem;
typedef vector<Elem> Row;
typedef vector<Row> Grid;
typedef vector<vector<bool>> MatBool;

bool canMove(Cell c1, Cell c2, const Grid& map){
    Elem blocked = map[c1.x][c1.y];
    for(int k = 0; k < blocked.size(); ++k){
        if(blocked[k].x == c2.x and blocked[k].y == c2.y)
            return false;
    }
    return true;
}

void visit(Cell v, int dx, int dy, const Grid& map,
           MatBool& visited, queue<Cell>& Q, int n, int m){
    Cell u;
    //calculate the position of the new cell to visit
    u.x = v.x + dx;
    u.y = v.y + dy;
    //the distance to u is the distance to v + 1
    u.d = v.d + 1;
    //test that u is inside the grid
    //that there is not a wall in the way from v to u
    //and that we have not visited u already
    if(not (u.x < 0 or u.x >= n or u.y < 0 or u.y >= m)
        and canMove(v, u, map) and not visited[u.x][u.y]){
        //put u in the queue to visit it in the future
        Q.push(u);
        //mark the position so we don't visit it again
        visited[u.x][u.y] = true;
    }
}

//continues on next page...
```

```
//...
int found(const Grid& map, MatBool& visited, int n, int m){
    //make a queue with the cells left to see
    queue<Cell> Q;
    Cell start;
    start.x = 0; start.y = 0; //start by the position 0,0
    Q.push(start);
    //mark the position so we don't see it again
    visited[start.x][start.y] = true;
    //while we still have cells left to see
    while(not Q.empty()){
        Cell v = Q.front(); Q.pop();
        //if we reach the end return the distance traveled
        if (v.x == n-1 and v.y == m-1){
            return v.d;
        }
        //if we don't, try to visit all neighbour cells
        visit(v, 1, 0, map, visited, Q, n, m);
        visit(v, -1, 0, map, visited, Q, n, m);
        visit(v, 0, -1, map, visited, Q, n, m);
        visit(v, 0, 1, map, visited, Q, n, m);
    }
    //no more cells to see and did not find the end
    return -1;
}

void fillGrid(Grid& map, int sX, int sY, int eX, int eY){
    Cell c1, c2;
    if(sY == eY){ //horizontal
        //sX and eX may come in any order, we need to make sure
        //that sX is the minimum of the two
        //we swap sX and eX only if they are not in order
        //http://www.cplusplus.com/reference/algorithm/minmax/
        //http://www.cplusplus.com/reference/tuple/tie/
        tie(sX, eX) = minmax({sX, eX});
        //the cells along the wall will be the ones at
        //sY-1 and eY (since sY and eY are equal)
        --sY;
        //block every pair of cells along the walls
        for(int x = sX; x < eX; ++x){
            c1.x = c2.x = x;
            c1.y = sY; c2.y = eY;
            map[x][sY].push_back(c2);
            map[x][eY].push_back(c1);
        }
    }
    //continues on next page...
```

```
//...
//the same for vertical
else{
    tie(sY, eY) = minmax({sY, eY});
    --sX;
    for(int y = sY; y < eY; ++y){
        c1.x = sX; c2.x = eX;
        c1.y = c2.y = y;
        map[sX][y].push_back(c2);
        map[eX][y].push_back(c1);
    }
}

int main (){
    int n,m, num_map;
    cin >> n >> m >> num_map;
    Grid map(n, Row(m, Elem(0)));
    for(int i = 0; i < num_map; ++i){
        int sX, sY, eX, eY;
        char u;
        //input the wall coordinates separated by a comma
        cin >> sX >> u >> sY >> u >> eX >> u >> eY;
        fillGrid(map, sX, sY, eX, eY);
    }

    MatBool visited(n, vector<bool>(m, false));
    int d = found(map, visited, n, m);
    cout << d << endl;
}
```

# 31 Keyboard

30 points

## Introduction

Nowadays, being proficient in English is mandatory. You know that, and that's why you've spent the last months studying hard for an official English exam. Since you're a computer programmer, you've chosen to do the computer based exam. The first part of the exam is the writing test. You arrive at the room, you sit in front of the computer and... OMG! The keyboard is not a standard one! It has several letters in each key.

You need to do the writing test with that strange keyboard. You have a text you have to write using that keyboard and you need to minimize the number of mistakes.

The length of the text you want to write and the text you will actually write must be the same. A mistake occurs when the letter of the  $i$ th position in the ideal text is not the same as the letter in the  $i$ th position in the real text.

For example, imagine you want to write HELLO\_WORLD and you have 5 different keys, with the following letters each key:

```
HEL
O_WOKLD
WOR
HE
LL
```

The best way to write HELLO WORLD is HE|LL|O\_WOKLD. In that case we've used the 2nd, 4th and 5th keys and we've done 1 mistake. We've written a K instead of an R.

## Input

Each case consist of the word  $W$  you want to write, then the number  $N$  of keys in the keyboard followed by  $N$  different sets of letters.

## Output

Print the minimum number of mistakes you'll do when you want to write the word  $W$  using this strange keyboard. Remember that  $W$  and the word you'll actually write must have the same length. If no combination of keys gives a word of the same length as  $W$ , print -1.

## Example 1

### Input

```
HELLO_WORLD  
5  
HEL  
O_WOKLD  
WOR  
HE  
LL
```

### Output

```
1
```

## Example 2

### Input

```
WE_ARE_THE_CHAMPIONS  
9  
ADKE_  
_  
ARE  
CHAM_PIO  
DDKK  
WEEEE  
W  
KLLLE  
IIWW
```

### Output

```
9
```



## Solutions

## Python3

```
import math

# read the data
WORD = input()
n = int(input())
setWords = []
for _ in range(n):
    setWords.append(input())

# sort setWords by length
setWords.sort(key=lambda x: len(x))
# each position "i" indicates the minimum number of
# mistakes you can do to write the subword WORD[:i]
mistakes = [0] + [float('inf')] * len(WORD)
for i in range(len(WORD)):
    maxLenSubWord = i+1
    minNumMistakes = float('inf')
    for w in setWords:
        # If w is larger than WORD[:maxLenSubWord] we stop
        # looking more words in setWords, because they will
        # be larger than the current word (w)
        if len(w) > maxLenSubWord:
            break

        subword = WORD[maxLenSubWord-len(w):maxLenSubWord]
        numMistakes = sum(w[j] != subword[j]\
                          for j in range(len(w)))\
                      + mistakes[maxLenSubWord-len(w)]
        minNumMistakes = min(minNumMistakes, numMistakes)
    mistakes[maxLenSubWord] = minNumMistakes
# print the answer
totalMinMistakes = mistakes[-1]
if totalMinMistakes < float('inf'):
    print(totalMinMistakes)
else:
    print(-1)
```

## C++

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

int getNumMistakes(string ideal, string real)
{
    int numErr = 0;
    for (int i = 0; i < ideal.size(); i++){
        if (ideal[i] != real[i]) numErr++;
    }
    return numErr++;
}

int main()
{
    string word;
    cin >> word;
    int numKeys = 0;
    int maxMistakes = word.size();
    cin >> numKeys;
    vector< string > keys(numKeys);
    vector< int > mistakes(word.size(), maxMistakes +1);
    for (int i = 0; i < numKeys; ++i) cin >> keys[i];

    for (int i = 0; i < word.size(); i++){
        int minErr = maxMistakes +1;
        for (int j = 0; j < keys.size(); j++){
            if (keys[j].size() > i+1) continue;

            string subWord = word.substr((i+1)-
keys[j].size(),
keys[j].size());
            int numMistakes = getNumMistakes(subWord,
keys[j]);

            int posPrevKey = i - keys[j].size();
            if ( posPrevKey >= 0)
                numMistakes += mistakes[posPrevKey];
            if (numMistakes < minErr) minErr = numMistakes;
        }
        mistakes[i] = minErr;
    }

    if (mistakes[mistakes.size() - 1] <= maxMistakes)
        cout << mistakes[mistakes.size() -1] << endl;
    else cout << -1 << endl;
}
```