1. Encoded Message

R	o	s	e	s
A	r	e	R	e
d	V	i	o	l
e	t	s	A	r
e	B	l	u	e

⇒

e	e	d	A	R
B	t	V	r	o
l	s	i	e	s
u	A	o	R	e
e	r	l	e	s

Alex wants to send a love poem to his girlfriend Bridget. Unfortunately, she has a nosy friend, Ellen, who might intercept his message and invade their privacy.

To prevent this, Alex has invented a scheme to make his missives indecipherable to Ellen. He arranges the letters into a square, which is rotated a quarter-turn clockwise, and then he puts the resulting letters on a single line again. (For simplicity's sake, Alex doesn't use whitespace or punctuation in his poems.)

For example, the text "RosesAreRedVioletsAreBlue" would be encoded as "eedARBtVrolsiesuAoReerles" using the following intermediate steps:

Ellen has intercepted some of Alex's messages but they make no sense to her. Can you write a program to help her decode them?
Input: On the first line one positive number: the number of test cases, at most 100. After that per test case: one line with an encoded message: a string consisting of upper-case and lower-case letters only. The length of the message is a square between 1 and 10000 characters.
Output: Per test case:

- one line with the original message.

Sample Input 1

```
3
RSTEEOTCP
eedARBtVrolsiesuAoReerles
EarSvyeqeBsuneMa
```

Sample Output 1

```
TOPSECRET
RosesAreRedVioletsAreBlue
SquaresMayBeEven
```

Looking at the pattern with a 9-character cyphertext mapping:

cyphertext position	0	1	2	3	4	5	6	7	8
plaintext position	6	3	0	7	4	1	8	5	2

This suggests the nested for-loop (where `n` is the square root of the cyphertext size):

```
for (int i = n - 1; i >= 0; i--)
    for (int j = 0; j < n; j++)
        cout << cypherText[n*j + i];
```

This would perhaps most simply be implemented like so:

```
#include <iostream>
#include <cmath>
#include <string>
using namespace std;
int main() {
    int N{ 0 }, n{ 0 };
    string cypherText, plainText;
    cin >> N;
    while (N--) {
        cin >> cypherText;
        n = sqrt(cypherText.size());
        for (int i = n - 1; i >= 0; i--) {
            for (int j = 0; j < n; j++) {
                cout << cypherText[n*j + i];
            }
        } cout << endl;
    }
}
```

But the problem, as stated, was to use pointers somehow. While there are a number of silly ways of incorporating pointers in irrelevant ways, the following captures the intent fairly well:

```
#include <iostream>
#include <cmath>
#include <string>
using namespace std;
#define MAXLENGTH 10000
int main() {
    int N{ 0 }, n{ 0 };
    string cypherText, plainText;
    char* cstr = new char[MAXLENGTH];
    cin >> N;
    cin.get();
    while (N--) {
        cin.getline(cstr, MAXLENGTH);
        n = sqrt(strlen(cstr));
        //from n-1, keep adding n until it gets to n-1 + n*(n-1), i.e,, n^2-1
        for (int i = n - 1; i >= 0; i--) {
            for (int j = 0; j < n; j++) {
                cout << *(cstr + n*j + i);
            }
        } cout << endl;
    }
}
```

Note the use of the `cin.get()` on line 11. Why? What happens if you remove it? What the heck is going on?!

2. Cryptographer's Conundrum

The walls of the corridors at the Theoretical Computer Science group (TCS) at KTH are all but covered with whiteboards. Some of the faculty members are cryptographers, and like to write cryptographic puzzles on the whiteboards. A new puzzle is added whenever someone discovers a solution to the previous one.

When Per walked in the corridor two weeks ago, he saw that the newest puzzle read "GuvfVfNGrfg". After arriving at his computer, he quickly figured out that this was a simple ROT13 encryption of "ThisIsATest".

The series of lousy puzzles continued next week, when a new puzzle read "VmkgdGFyIHPDpGtlcmhldGVuIHDDpSBzdMO2cnN0YSBhbGx2YXIK". This was just base64-encoded text! "Enough with these pranks", Per thought; "I'm going to show you!"

Now Per has come up with a secret plan: every day he will erase one letter of the cipher text and replace it with a different letter, so that, in the end, the whole text reads "PerPerPerPerPerPerPer". Since Per will change one letter each day, he hopes that people will not notice.

Per would like to know how many days it will take to transform a given cipher text into a text only containing his name, assuming he substitutes one letter each day. You may assume that the length of the original cipher text is a multiple of 3.

For simplicity, you can ignore the case of the letters, and instead assume that all letters are upper-case.

Input

The first and only line of input contains the cipher text on the whiteboard. It consists of at most 300 upper-case characters, and its length is a multiple of 3.

Output

Output the number of days needed to change the cipher text to a string containing only Per's name.

Sample Input 1	Sample Output 1
SECRET	4

Here's a solution that uses pointer arithmetic:

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

int main() {
    int N = 0, n = 0;
    char per[4] = "PER";
    char* cstr = new char[301];
    cin.getline(cstr, 301);
    n = strlen(cstr);
    for (int i = 0; i < n; ++i, ++cstr)
        if (*cstr != per[i % 3])
            ++N;
    cout << N;
}
```

Note the two ways to create a c-string used here: `char per[4] = "PER";` creates a string literal whose fourth character will be the NUL character, `'\0'`, while `char* cstr = new char[301];` allocates enough memory in the free store for 301 characters (that would be 301 bytes, since each character requires one byte) and assigns the address in memory of the first of these the variable `cstr` which is of type pointer to `char`.

We use `cin.getline(cstr,301);` to input a line from the console into `cstr`, allowing for this line to be at most 300 characters long (and allocating an extra char for the NUL which will be affixed at the end.)

The `strlen()` function is in the `cstring` library, which is automatically included in Visual Studio, but not in Kattis, so be sure to `include cstring` if you're submitting to Kattis. This will return the number of characters preceding the NUL, which will be 6, in the case of "`SECRET`".

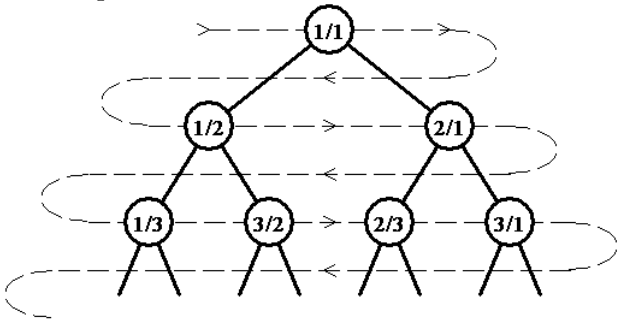
The `for` loop uses a basic counter `i` and the character pointer `cstr` each character in the given string, checking to see if the character is P, E or R in the cyclically 0, 1 or 2 position. Note that we dereference the pointer to the input string with `*cstr`, which will be the first character in c-string memory allocated at that position, and as the for loop increments `i`, it also increments the pointer `cstr`.

3. A Rational Sequence

A sequence of positive rational numbers is defined as follows:
An infinite full binary tree labeled by positive rational numbers is defined by:

- The label of the root is 1/1
- The left child of label p/q is $p/(p+q)$
- The right child of label p/q is $(p+q)/q$.

The top of the tree is shown below:



The sequence is defined by doing a level order (breadth first) traversal of the tree (indicated by the light dashed line). So that: $F(1) = 1/1, F(2) = 1/2, F(3) = 2/1, F(4) = 1/3, F(5) = 3/2, F(6) = 2/3 \ldots$.

Write a program which finds the value of n for which F(n) is p/q for inputs p and q.
A few key observations:
1) you can determin whether a node is a left child or a right child by whether or not its numerator is, respectively, less or more than the denominator.
2) In retracing your path from N/M to 1/1 up through the tree to the root, you'll have a sequence of Right/Left moves where the number of moves give your depth in the tree and the number of times you're a right child tells what to add to that level to get your position.

Recursively speaking, (explitives deleted) this means multiplying the position variable by 2 everytime you go up one and adding 1 every time you're a right child going up. That's actually then very simple!

However, as often happens with a first go, my first solution involved the creation of a `BTPos` class with a `righLefts` string to record left/right children status as a path is wended from the target node to the root, replete with `getRightLefts()` member function and a `getN()` helper function!

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

class BTPos {
    int p, q, level;
    string rightLefts;

public:
    BTPos(int p, int q) : p(p), q(q) {}
    int getlevel(int p, int q) {
        level = 0;
        if (p == 1 && q == 1) return 0;
        else if (p > q) {
            cout << "\np = " << p << "q = " << q;
            return 1+getlevel(p - q, q);
        }
        else {
            cout << "\np = " << p << "q = " << q;
            return 1+getlevel(p, q - p);
        }
    }

    string getRightLefts(int p, int q) {
        if (p == 1 && q == 1) return "";
        else if (p > q) {
            cout << "\np = " << p << "q = " << q;
            return  getRightLefts(p - q, q)+'R';
        }
        else {
            cout << "\np = " << p << "q = " << q;
            return getRightLefts(p, q - p)+'L';
        }
    }
};

int getN(string s, int level) {
    int value{ 0 };
    int pow2 = 1;
    for (int i = 0; i < s.size(); ++i) {
        if (s[i] == 'R') {
            value += 1 * pow2;

        }
        pow2 *= 2;
    }
    return pow2 + value;
}
```

```
int main() {
    int P, p, q, k, level;
    cin >> P;
    char c;
    while (P--) {
        cin >> k >> p >> c >> q;
        BTPos btp(p, q);
        cout << "\nlevel = " << btp.getlevel(p,q)
            << "\nrightLefts = " << btp.getRightLefts(p,q);
        cout << "\nN = " << getN(btp.getRightLefts(p, q), btp.getlevel(p, q));
    }
    cin.ignore();
    cin.get();
}
```

After some reflection, we (in class) were able to write a much simpler (more purposed) program that used only 30 lines of code, instead of 63 lines above:

```
// Kattis Contest Problem: A Rational Sequence 2
#include <iostream>
#include <fstream>
using namespace std;

int ftn(int p, int q);

int main() {
    int k, P, p, q;
    char c;
    fstream ifs("sample001.in", fstream::in);
    fstream ofs("check.ans", fstream::out);
    ifs >> P;

    while (P--) {
        ifs >> k >> p >> c >> q;
        ofs << k << " " << ftn(p, q);
    }
}

int ftn(int p, int q) {
    //base case
    if (p == 1 && q == 1) return 1;
    if (p > q) {
        return 1 + 2 * ftn(p - q, q);
    }
    else {
        return 2 * ftn(p, q - p);
    }
}
```

4. Collatz Conjecture

Here the challenge was to write a recursive function that would accept an intitial positive integer and then iterate

$$x_{n+1} = \begin{cases} x_n/2 & : \texttt{if } x_n \texttt{ is even} \\ 3 \cdot x_n + 1 & : \texttt{if } x_n \texttt{ is odd} \end{cases}$$

with the expectation that the conjecture is true: it will end with 4,2,1, repeating.

This was meant to be easy, but it kept many of you busy for a while, nonetheless. Here's a solution:

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

int collatz(int N) {
    if (N == 1) return 1;
    else if (N % 2 == 0) {
        cout << N << " ";
        return collatz(N / 2);
    }
    else {
        cout << N << " ";
        return collatz(3 * N + 1);
    }
}

int main() {
    int N;
    cout << "\nEnter a positive number for N:\n";
    while (cin >> N) {
        cout << collatz(N);
    }
}
```