Poly SQRT

In this problem, you are given the coefficients of a polynomial $P(x)$ with integer coefficients and need to determine whether there is a polynomial $Q(x)$ with integer coefficients such that $P(x) = Q(x) * Q(x)$.

Input to your program will be specifications of polynomials over the integers of degree* ≤ 100. For each polynomial, the first line will indicate the degree n of a polynomial $P(x)$. The following $n+1$ lines will give the coefficients of the powers of x in $P(x)$ in descending order of degree. For example,

```
2
3
-7
0
```

specifies the polynomial $3x^2 - 7x$.

Output for each polynomial $P(x)$ should indicate the (integer) coefficients of some $Q(x)$ such that $P(x) = Q(x) * Q(x)$, one integer per line, or else the phrase

"`No square root among integer polynomials.`"

There should be one line skipped in the output between polynomials.

Sample Input	Sample Output

`2`
`4`
`-12`
`9`
`0`
`3`
`2`
`4`
`0`
`0`
`3`
`1`
`1`
`1`
`1`
	`2`
	`-3`
	
	`No square root among integer polynomials.`
	
	`2`
	`0`
	
	`No square root among integer polynomials.`

*The *degree* of a polynomial in x is the highest power of x that occurs with a non-zero coefficient.

Voluminous Vocabulary

The languages developed throughout the history of the Vulcan planetary system reflected the inherent efficiency of the Vulcan mind. Each language made exhaustive use of its alphabet by assigning a word to every combination of letters up to a maximum word length. For example, the primitive `XYX`ians had just a two-letter alphabet, {`X`, `Y`} and, since their words have length ≤ 3, the entire `XYX`ian dictionary consisted of the 14 words

`X,XX,XXX,XXY,XY,XYX,XYY,Y,XY,YXX,YXY,YY,YYX,YYY`

(listed here in dictionary order). Of course, through succeeding millenia, the languages of the advanced planets needed larger alphabets and word lengths to accommodate the enormous Vulcan vocabularies.

In this problem, you will be given a Vulcan language via its alphabet and the upper bound to word lengths, along with a word in the language. You need then determine the position of this word in the dictionary.

Input to your program will be specifications of languages. For each language, the first line will list the the number (≤ 12) of letters in the alphabet, followed after a space by the maximum word length (≤ 12). The second line will list the alphabet[†] itself (in its own alphabetical order, separated by single spaces). Finally, the third line for each language will contain a word $\mathcal{W}$ in the language.

Output for each such language should be a line with the integer n such that $\mathcal{W}$ is the n^{th} word in the dictionary.

Sample Input	Sample Output

`3 2`
`Y O U`
`UO`
`2 4`
`O B`
`BOO`
	`11`
	`18`

[†]For the convenience of Terran linguists, the letters of the Vulcan alphabets have been transliterated to English characters but they are considered in their native Vulcan alphabetical order.

MC/DC Test Vector Generation

This problem is based on the "modified condition/decision coverage" (MC/DC) criterion, which is required by U.S. and European agencies for certification of safety-critical flight control software. Briefly, the MC/DC criterion requires testing with data that shows that each elementary term (e.g., comparison) of a predicate be used to control the outcome of each test (e.g., `if` statement).

For example, if the program contained a statement like

```
if (x > 30 && x < y) { ...
```

then the following assignments of truth values would satisfy the MC/DC criterion:

	`x > 30`	`x < y`	Result
Case 1:	True	True	True
Case 2:	False	—	False
Case 3:	True	False	False

Cases 1 and 2 show the effect of the first term, `x > 30`. Cases 1 and 3 show the effect of the second term, `x < y`. Note that in Case 2, the second term of the predicate is not evaluated, because of short-circuit evaluation. We call it a "don't-care" value. For purposes of the MC/DC criterion we consider that the value of that term to be unchanged between Case 1 and Case 2 because a don't-care value could be either True or False.

For the contest, the problem has been considerably simplified. Each elementary term of a predicate is represented by a single alphabetic character, and the boolean expression is represented in postfix syntax with one-character operators for easy parsing. Thus the above predicate `(x > 30 && x < y)` would appear as `ab&`.

Your program will read lines of text from the standard input stream. Each line of text will contain a boolean expression in postfix syntax. The program should print on the standard output stream assignments of truth values to terms in the boolean expression, and the resulting truth value of the expression.

Each input expression is a single line of text, no more than 30 characters in length. The non-blank characters in the line are interpreted as follows:

a-z	An alphabetic character represents an elementary term in the predicate. No alphabetic character will appear more than once in the input line.
`&`	Conjunction ("and"), with short-circuit evaluation.
`|`	Disjunction ("or"), with short-circuit evaluation.
`!`	Negation ("not").

The line of text will be a single syntactically valid postfix expression. For example, the Java predicate `((x > y && y > 0) || ! (y < 100))` would appear as `ab&c!|`.

If an input expression contains N alphabetic characters, then the output consists of exactly $N + 1$ lines of text, each line containing containing exactly $N + 1$ characters. Each character of the output is either the upper-case letter "T" denoting the boolean value "true," the upper-case letter "F" denoting the boolean value "false," or the character "*" denoting a don't-care value.

The i^{th} character of each output line, $i \leq N$, represents an assignment of a boolean value or a "don't care" value to the i^{th} boolean expression. A don't-care value must appear if and only if the corresponding term would not be evaluated in a short-circuit evaluation. The $(N+1)^{\text{st}}$ character of each output line must represent the boolean result of evaluating the expression with the assigned truth values.

For each column i, $1 \leq i \leq N+1$, the output must contain two lines m and n with the following properties:

- The i^{th} position of line m differs from the i^{th} position of line n.
- The $(N+1)^{\text{st}}$ position of line m differs from the $(N+1)^{\text{st}}$ position of line n.
- For every other position j, $j \neq i$ and $1 \leq j \leq N$, either the j^{th} position of line m is the same as the j^{th} position of line n, or else the j^{th} position of at least one of the two lines is the "don't-care" symbol "*."

These properties correspond to the MC/DC test coverage requirement that lines m and n represent truth assignments in which the value of term i controls the value of the boolean expression.

The output from successive expressions should be separated by a single blank line.

Sample Input	Sample Output
`ab|`	`T*T`
`ab&c!|`	`FTT`
`ab&cd&ef&||`	`FFF`
	`TT*T`
	`TFFT`
	`TFTF`
	`F*TF`
	`TT****T`
	`TFTT**T`
	`TFTFTTT`
	`TFTFTFF`
	`F*TFTFF`
	`TFF*TFF`
	`TFTFF*F`

Example 1: **(a or b)**

Lines 1 and 2 show the effect of a; lines 2 and 3 show the effect of b.

Example 2: **((a and b) or not c)**

Lines 1 and 4 show the effect of a. Lines 1 and 2 show the effect of b. Lines 2 and 3 show the effect of c.

Example 3: **((a and b) or (c and d) or (e and f))**

Note that the correct output is not unique: Many different sets of truth assignments could have the same properties.

λ.COM (Java Version)

Having studied the λ-calculus, the president of a ".com" startup has stored sensitive financial numbers in instances of the Java class `SecretInfo` defined below:

```
interface Encoded {
    Object decode (Decoder f, Object x);
}

interface Decoder {
    Object decode (Object x);
}

class SecretInfo implements Encoded {
    private int n; // the secret

    SecretInfo (int n) { this.n = n; }

    public Object decode (Decoder f, Object x) {
        Object current = x;
        for (int i=0; i<n; i++) current = f.decode(current);
        return current;
    }
}
```

As the code shows, the only way to get any information about the secret number is to use the `decode` method. If a `SecretInfo` object is hiding number n, then its decode method will apply the `Decoder` n times.

A new employee in the company has just figured out a way to make millions. She comes to you for help. All she needs is for you to complete the implementation of the method `decode` below:

```
  class CompromisedInfo implements Encoded {
    private Encoded originalInfo;

    CompromisedInfo (Encoded originalInfo) {
      this.originalInfo = originalInfo;
    }

    public Object decode (Decoder f, Object x) {
      ...
    }
  }
```

If the `originalInfo` is hiding a number n, then your `decode` method should arrange for the `Decoder` to be applied $3n$ times instead of n times.

To solve the problem, turn in the class `CompromisedInfo` and any other help classes you need.

λ.COM (C++ Version)

Having studied the λ-calculus, the president of a ".com" startup has stored sensitive financial numbers in instances of the C++ class `SecretInfo` defined below:

```
class Decoder {
    public:
    virtual void* decode (void* x) { return (void*)0; }
};

class Encoded {
    public:
    virtual void* decode (Decoder* f, void* x) {
        return (void*)0;
    }
};

class SecretInfo : public Encoded {
    private:
    int n; // the secret

    public:
    SecretInfo (int n) { this->n = n; }

    virtual void* decode (Decoder* f, void* x) {
        void* current = x;
        for (int i=0; i<n; i++) current = f->decode(current);
        return current;
    }
};
```

As the code shows, the only way to get any information about the secret number is to use the `decode` method. If a `SecretInfo` object is hiding number n, then its decode method will apply the `Decoder` n times.

A new employee in the company has just figured out a way to make millions. She comes to you for help. All she needs is for you to complete the implementation of the method `decode` below:

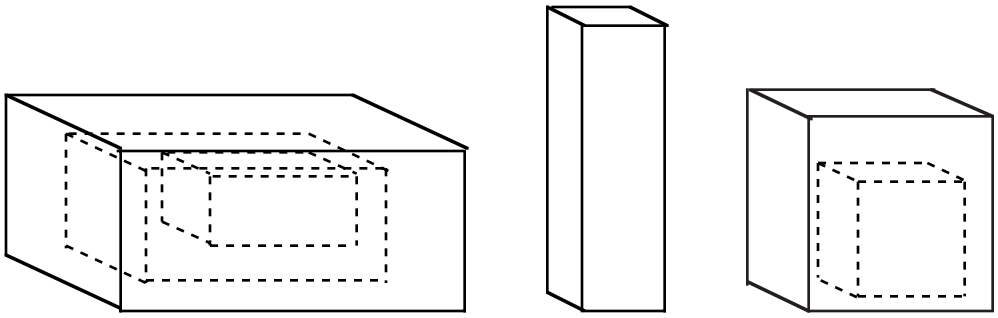
```
class CompromisedInfo : public Encoded {
    private:
    Encoded* originalInfo;

    public:
    CompromisedInfo (Encoded* originalInfo) {
        this->originalInfo = originalInfo;
    }

    virtual void* decode (Decoder* f, void* x) {
        ...
    }
};
```

If the `originalInfo` is hiding a number n, then your `decode` method should arrange for the `Decoder` to be applied $3n$ times instead of n times.

To solve the problem, turn in the class `CompromisedInfo` and any other help classes you need.

Bargain Boxing

Confederated Express charges a fixed rate per package for shipping packages up to 36 inches in their maximum dimension. BigTime Boxes, Inc., uses ConEx to send their elite boxes. Since the rectangular boxes are themselves empty, they can save considerably by nesting boxes inside other boxes when feasible.



Box B will fit inside box B' if, *after suitable rotation*, the length, width, and height of B are, respectively, at least *one inch less than* the length, width, and height of B. For example, a $5'' \times 3'' \times 6''$ box will fit inside a $4'' \times 7'' \times 6''$ box (since $3 < 4$, $5 < 6$, and $6 < 7$.

In this problem, you are given the dimensions of the boxes in a planned shipment by BigTime Boxes. You need to determine the minimum number of packages that BigTime will turn over to ConEx.

Input to your program will be specifications of shipments. The first line for each shipment will be the number of boxes n in the shipment ($1 \leq n \leq 25$). The following n lines will each consist of a triple of integers $l\ w\ h$ (with one space separation) with $1 \leq l, w, h \leq 36$, indicating the length, width, and height of a box in inches.

Output for each shipment should be the minimum number of nested packages that will accommodate all the boxes in the shipment.

Sample Input

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

Sample Output

```
1
3
```