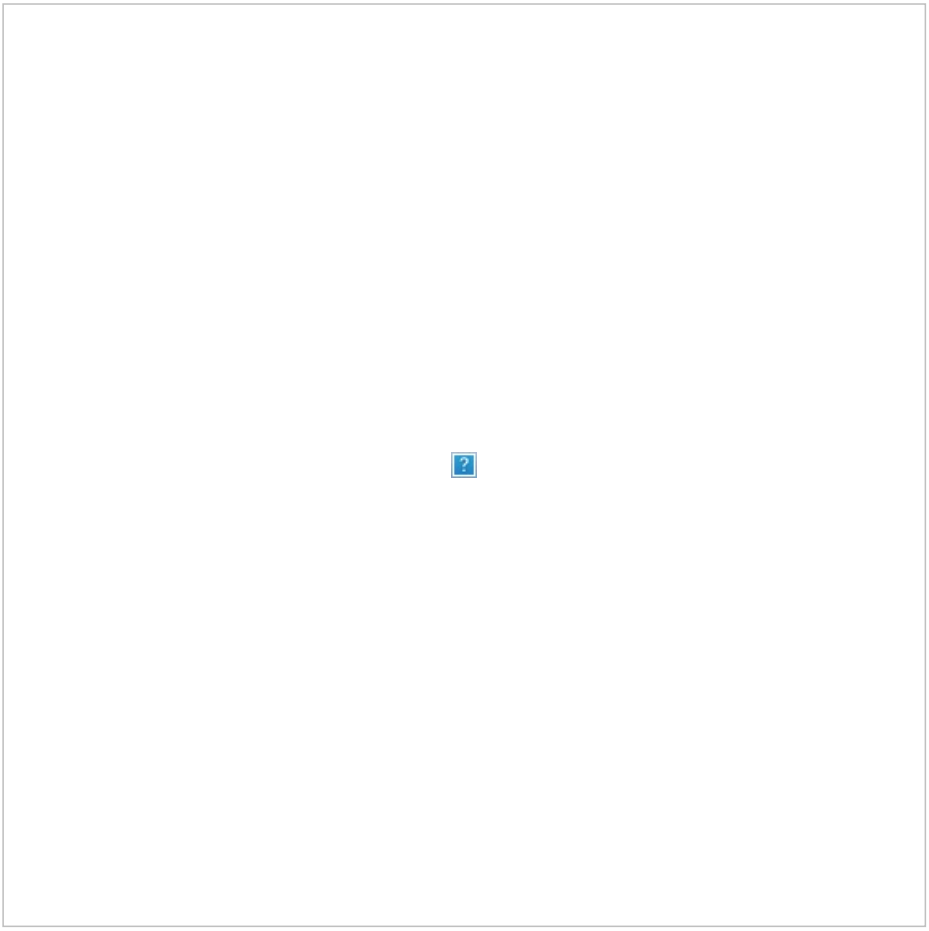
English version | [ver versão em português]

[PI052] - Reverse Polish Notation

The Problem



Back in elementary school, you learned that different arithmetic operators have different precedences. This is true in every country. For example, in English, there's a mnemonic to remember the order: ***"Please Excuse My Dear Aunt Sally"*** – representing the order: **P**arentheses, **E**xponents, **M**ultiplication, **D**ivision, **A**ddition, and **S**ubtraction.

For instance, the expression:

```
1 + 2 * 3 = ?
```

results in `7` because multiplication takes precedence over addition.

If we wanted to perform the addition first, we would need parentheses:

```
(1 + 2) * 3 = ?
```

However, in the early days of electronic calculators, parsing expressions with precedence and parentheses proved complicated.

To address this, the Polish mathematician **Jan Łukasiewicz** introduced a notation in the 1920s that made parentheses unnecessary, ensuring operations would be evaluated in the desired order. In his system, operators are written **before** their operands.

Later, in the 1950s, computer scientist **Charles L. Hamblin** proposed placing operators **after** the operands instead. This variant, which became very popular due to its ease of implementation using stacks, is known as **Reverse Polish Notation (RPN)**, or *postfix notation*. In RPN, even complex expressions can be written without parentheses.

Infix Notation	Reverse Polish Notation (RPN)
1 + 1	1 1 +
3 * (4 + 5)	3 4 5 + *
3 + 4 * 5	3 4 5 * +

Your task is to write a program that evaluates expressions written in RPN. A standard evaluation algorithm works as follows:

- Create an empty stack.
- Process the expression from left to right:
  - If the element is a number, push it onto the stack.
  - If it is an operator, pop the top two numbers from the stack, apply the operator, and push the result back onto the stack.
- At the end, there should be exactly one value on the stack — the result of the expression.
- If at any point this rule is violated (e.g., trying to pop from an empty stack), the expression is considered invalid.

Input

The first line contains an integer **N**, the number of expressions to evaluate.

The following **N** lines each contain a single RPN expression, consisting of:

- Integers (positive or negative, always valid as stand-alone tokens)
- Operators: `+`, `-`, `*`, `/`

You may assume that:

- All numbers are integers.
- Intermediate results are always integers.
- Only the four basic operations are used.
- The length of an expression, in characters, is less than 10,000.

Output

For each expression, print a single line:

- The result of the expression, if it is valid.
- **`Expressao Incorreta`** (*`Incorrect Expression`*, if the expression is malformed (e.g., stack underflow, leftover items, or division by zero).

Exemplo de input/output

Input	Output
6 1 1 + 3 4 5 + * 3 4 5 * + 1 + 1 1 1 + 2 3 8 2 / - 1 + *	2 27 23 Expressao Incorreta Expressao Incorreta 0

Versão em Português | [see english version]

PI052 - Notação Polaca Invertida

O problema

Quando andaste na escola primária, aprendeste que os diferentes operadores aritméticos têm diferentes precedências. Isso acontece em todos os países, e por exemplo os ingleses usam a mnemónica: *"**P**lease **E**xcuse **M**y **D**ear **A**unt **S**ally"*, que indica a ordem dos operadores (**P**arênteses, **E**xponenciação, **M**ultiplicação, **D**ivisão, **A**dição, **S**ubtração).

Por exemplo, a expressão:

```
1 + 2 x 3 = ?
```

dá como resultado 7, pois o operador multiplicação tem precedência.

Se quisermos avaliar primeiro a soma a expressão tem de ser:

```
(1 + 2) x 3 = ?
```

No entanto, nos tempos iniciais das calculadoras electrónicas, revelou-se complicado analisar este tipo de expressões.

Por isso mesmo, em 1920, o matemático polaco **Jan Lukasiewicz** criou uma notação que tornava desnecessário o uso de parênteses, garantindo sempre que as operações eram efectuadas como desejado. A notação consistia basicamente em escrever os operadores antes dos números e não no meio deles.

Já em 1950, o *computer scientist* **Charles L. Hamblin** propôs um esquema onde os operadores apareciam a seguir aos números, em vez de ser antes. Esta notação acabou por ser muito usada, devido entre outras coisas à sua fácil implementação num sistema automático usando uma pilha, e ficou conhecida como **Notação Polaca Invertida (RPN)** (ou *postfix*). Em RPN, uma expressão arbitrariamente complexa pode ser escrita sem o uso de parênteses.

Notação Normal	**Notação Polaca Invertida (RPN)**
1 + 1	1 1 +
3 * (4 + 5)	3 4 5 + *
3 + 4 * 5	3 4 5 * +

A tua tarefa é criar um programa capaz de calcular o valor final de expressões dadas em RPN, sabendo que um algoritmo para as analisar é:

- Criar uma pilha
- Ir da esquerda para a direita lendo elemento a elemento e:
  - Se o elemento for um número, inseri-lo na pilha
  - Se for um operador, retirar dois valores da pilha, aplicar esse operador, e inserir o resultado na pilha
- No final, fica apenas um valor na pilha, que é o resultado da expressão
- Se a pilha não corresponder a alguma das coisas que foi dita atrás (por exemplo, ficar vazia), então a expressão é incorrecta.

Input

A primeira linha contém um número **N**, indicando o número de expressões a analisar.

As seguintes **N** linhas contêm expressões RPN, contendo

- Dígitos, representando números inteiro
- Caracteres, representando operações (+, -, * , /)

Podes assumir que os números são sempre inteiros, que os cálculos intermédios vão dar sempre números inteiros, e que só aparecem os quatro tipos básicos de operações atrás citados. O comprimento de uma expressão não excede os 10,000 caracteres.

Output

Uma linha para cada expressão, indicando o resultado final caso a expressão seja correcta, ou **`Expressao Incorreta`**, caso contrário.

Exemplo de input/output

Input	Output
6 1 1 + 3 4 5 + * 3 4 5 * + 1 + 1 1 1 + 2 3 8 2 / - 1 + *	2 27 23 Expressao Incorreta Expressao Incorreta 0
