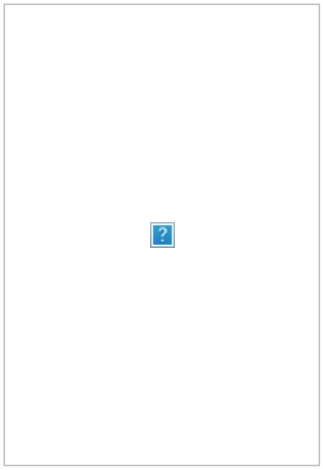
English version | [ver versão em português]

PI048 - Between Brackets



Joanna was doing a math exercise when she came across the following expression:

```
( 1 + ( 2 + 3 * [ 4 + 5 * ( 6 * ( 7 ) ) ] ) )
```

"So many brackets!" she thought. And before even starting the problem, another thought came to mind: "What if the teacher made a mistake and didn't place the brackets correctly? Then it's not even worth solving." Joanna is quite the procrastinator.

Since Joanna trusts you, she's asking for your help: write a program that checks whether an expression has well-formed brackets so she can confidently get to work.

Your task is to build a bracket validator. It must determine if the brackets in a given mathematical expression are well-formed and, if not, report the first position where an error occurs. You don't need to worry about the actual mathematics — just the brackets.

Input

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

The following **N** lines contain mathematical expressions with:

- Digits, representing integer numbers
- Operators: **+**, **-**, *****, or **/**
- Round and square brackets: **(**, **)**, **[**, and **]**
- Spaces between all elements of the expression

You may assume the expressions are reasonably sized. Every component is separated by spaces, and there are no characters other than the ones described above.

Output

For each expression, print a single line with one of the following messages (in Portuguese):

- `Expressao bem formada` (*Well-formed expression*) – if all brackets are matched and properly nested
- `erro na posicao` *`POS`* (*Error at position POS*) – if a mismatched closing bracket is found at character position *POS*
- `Ficam parenteses por fechar` (*Unclosed brackets remain*) – if there are unmatched opening brackets by the end of the line

Note: the position refers to the character index in the original line (starting at 0). For example, in the line `( 1 + 2 )`, the character `(` is at position 0, `+` is at position 4, and `)` is at position 8.

All non-bracket characters can be ignored — for this problem, only brackets matter.

Example Input/Output

Input	Output
5 (1) (1 + 2)) [( ]) (1 + (2) (1 + (2 + 3 * (4 + 5 * (6 * (7)))))	Expressao bem formada Erro na posicao 10 Erro na posicao 4 Ficam parenteses por fechar Expressao bem formada

Suggestion: Use this code to handle the input and just implement the `verify()` function and/or other functions that you may find necessary. Submit the full program.

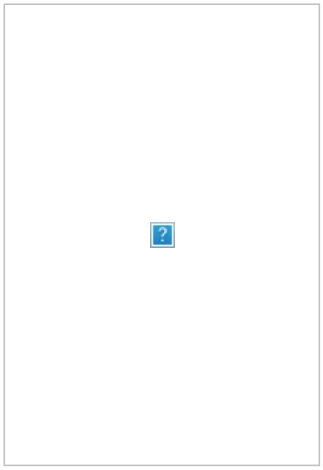
```c
#include <stdio.h>
#include <string.h>

#define MAX 200
#define SKIPEOL {while (getchar()!='\n') ;}
```

```c
void verify(char *line);

int main() {
  int ncases;
  char line[MAX];

  scanf("%d",&ncases);
  SKIPEOL;
  for (int c=0; c<ncases; c++) {
    fgets(line,sizeof(line),stdin);
    verify(line);
  }
}
```

Versão em Português | [see english version]

PI048 - Entre Parênteses

O problema



A Joana estava a fazer um exercício de matemática quando se deparou com a seguinte expressão:

```
( 1 + ( 2 + 3 * [ 4 + 5 * ( 6 * ( 7 ) ) ] ) )
```

"Tantos parênteses!", pensou ela. E antes de começar a fazer exercício pensou também: "E se o professor pensou mal antes de começar a fazer o exercício e não colocou bem os parênteses. Se assim for nem vale a pena fazer o exerício". É tão preguiçosa esta Joana.

Como a Joana acredita em ti, tens de tentar fazer um programa que indique que a expressão matemática está mesmo bem formada, para que ela tenha vontade de fazer o exercício.

A tua tarefa é portanto fazer um analisador de expressões que ligue apenas aos parênteses. Tens de conseguir dizer se os parênteses estão bem formados, e caso contrário, indicar qual a posição onde foi encontrado o primeiro erro. E não é preciso explicar quando uma expressão com parênteses está bem formada, pois não?

Input

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

As seguintes **N** linhas definem expressões matemáticas contendo:

- Dígitos, representando números inteiros
- Caracteres, representando operações: **+**, **-**, ***** ou **/**
- Parênteses curvos ou rectos: **(**, **)**, **[** ou **]**
- Espaços separando as várias partes da expressão

Não podes assumir nenhum tamanho máximo para a expressão, mas é garantido que existem espaços a separar todas as componentes da expressão.

Output

Uma linha para cada expressão, indicando **`Expressao bem formada`** caso todos os parenteses estejam bem emparelhados, **`Erro na posicao`** ***`POS`***, caso seja descoberto um parênteses que não poderia aparecer naquele sítio (onde **POS** indica a posição na linha, começando a contar de zero, desse mesmo parênteses que está incorrecto) ou **`Ficam parenteses por fechar`** caso chegando ao fim da expressão fiquem ainda parênteses por fechar.

Note-se que por posição entende-se o número de caracteres até chegar ao sítio em causa. Por exemplo, na linha "`( 1 + 2 )`", o `(` está na posicao 0, o `+` na posição 4 e o `)` na posição 8.

Lembra-te que podes ignorar todo o conteúdo da expressão que não sejam parenteses, pois para este problema em particular, é irrelevante.

Exemplo de input/output

Input	Output

5 (1) (1 + 2)) [( ]) (1 + (2) (1 + (2 + 3 * (4 + 5 * (6 * (7)))))	Expressao bem formada Erro na posicao 10 Erro na posicao 4 Ficam parenteses por fechar Expressao bem formada

Sugestão: Use o código seguinte para a leitura e implemente apenas a função `verify()` e outras que considere necessárias. Submeta o programa completo.

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

#define MAX 200
#define SKIPEOL {while (getchar()!='\n') ;}

void verify(char *line);

int main() {
  int ncases;
  char line[MAX];

  scanf("%d",&ncases);
  SKIPEOL;
  for (int c=0; c<ncases; c++) {
    fgets(line,sizeof(line),stdin);
    verify(line);
  }
}
```
