Paréntesis equilibrados

Decimos que una secuencia de caracteres que, entre otros caracteres, contiene paréntesis, llaves y corchetes abiertos y cerrados, está *equilibrada* si de cada uno de ellos tiene tantos abiertos como cerrados y si cada vez que aparece uno cerrado, el último que apareció aún no emparejado fue su correspondiente abierto.

Entrada

La entrada está formada por una serie de casos de prueba, cada uno en una línea. Cada caso consiste en una secuencia de caracteres.

Salida

Para cada caso de prueba se escribirá en una línea la palabra `SI` si la secuencia de caracteres está equilibrada y `NO` en caso contrario.

Entrada de ejemplo

```
Hola, me llamo Jose (Pepe para los amigos).
Yo soy Francisco (o Paco]
{((())}[]
]
```

Salida de ejemplo

```
SI
NO
SI
NO
```

Autor: Alberto Verdejo.

```
//
//  stack_eda.h
//
//  Implementación del TAD pila con array dinámico
//
//  Estructuras de Datos y Algoritmos
//  Facultad de Informática
//  Universidad Complutense de Madrid









                    ______        ____
    size_t nelems;

    // tamaño del array
    size_t capacidad;

    // puntero al array que contiene los datos (redimensionable)
    T * array;

public:

    // constructor: pila vacía
    stack() : nelems(0), capacidad(TAM_INICIAL), array(new T[capacidad]) {}

    // destructor
    ~stack() {
        libera();
    }

    // constructor por copia
```

```
    void push(T const& elem) {
        if (nelems == capacidad)
            amplia();
        array[nelems] = elem;
        ++nelems;
    }

    // consultar la cima
    T const& top() const {
        if (empty())
            throw std::domain_error("la pila vacia no tiene cima");
        return array[nelems - 1];
    }

    // desapilar el elemento en la cima
    void pop() {
        if (empty())
            throw std::domain_error("desapilando de la pila vacia");
        --nelems;
    }

    // consultar si la pila está vacía
    bool empty() const {
        return nelems == 0;
    }

    // consultar el tamaño de la pila
    size_t size() const {
        return nelems;
    }

protected:

    void libera() {
        delete[] array;
        array = nullptr;
    }

    // this está sin inicializar
    void copia(stack const& other) {
        capacidad = other.nelems;
        nelems = other.nelems;
        array = new T[capacidad];
        for (size_t i = 0; i < nelems; ++i)
            array[i] = other.array[i];
    }

    void amplia() {
        T * viejo = array;
        capacidad *= 2;
        array = new T[capacidad];
        for (size_t i = 0; i < nelems; ++i)
            array[i] = std::move(viejo[i]);
        delete[] viejo;
    }
};
```

```
#endif // stack_eda_h
```

```cpp
#include "stack_eda.h"

#include <iostream>
#include <fstream>
#include <string>

bool resuelveCaso(){
    std::string cadena;
    stack<char> pila;

    std::getline(std::cin, cadena);
    if (!std::cin){
        return false;
    }

    int i = 0;
    while (i < cadena.size()){
        //Si el caracter que viene es uno de apertura se incluye directamente en la pila
        if (cadena[i] == '(' || cadena[i] == '[' || cadena[i] == '{'){
            pila.push(cadena[i]);
        }
        //Si el caracter que viene es uno que cierra debemos comprobar si el anterior
        //es uno del mismo tipo que abre, para poder quitarlo de la pila
        else if (cadena[i] == ')'){
            if (!pila.empty() && pila.top() == '('){
                pila.pop();
            }
            else {
                pila.push(cadena[i]);
            }
        }
        else if (cadena[i] == ']'){
            if (!pila.empty() && pila.top() == '['){
                pila.pop();
            }
            else {
                pila.push(cadena[i]);
            }
        }
        else if(cadena[i] == '}'){
            if (!pila.empty() && pila.top() == '{'){
                pila.pop();
            }
            else {
                pila.push(cadena[i]);
            }
        }
        i++;
    }

    if (pila.empty()){
        std::cout << "SI" << std::endl;
    }
    else{
        std::cout << "NO" << std::endl;
    }
```

```
    return true;
}

int main() {

    // ajuste para que cin extraiga directamente de un fichero
#ifndef DOMJUDGE
    std::ifstream in("casos.txt");
    auto cinbuf = std::cin.rdbuf(in.rdbuf());
#endif

    while (resuelveCaso());
    system("PAUSE");
    // restablecimiento de cin
#ifndef DOMJUDGE
    std::cin.rdbuf(cinbuf);
#endif

    return 0;
}
```