Ejercicios del profe: Pilas & Colas 2025-1

Ludwig Alvarado Becerra

6 de junio de 2025 — Universidad Jorge Tadeo Lozano - Semillero de Programación Competitiva

UTADEO

Problema 1

Problema 1

Dada una cadena, **exp**, que puede constar de paréntesis de apertura y cierre, su tarea es comprobar si la cadena contiene paréntesis válidos.
Las condiciones para validar son las siguientes:

- Todo paréntesis de apertura debe cerrarse con el mismo tipo de paréntesis.
  - Por lo tanto, y como ejemplo, las cadenas `{)` y `[ ( ])` no son válidas.
- Cada paréntesis de apertura debe cerrarse en el orden correcto.
  - Por lo tanto, y como ejemplo, `)(` y `()(()` no son válidos.

Restricciones

- `1 <= exp.length <= 10^3`
- La cadena solo contendrá los siguientes caracteres: `(, ), [, ], {` y `}`

Problema 1 | Tests

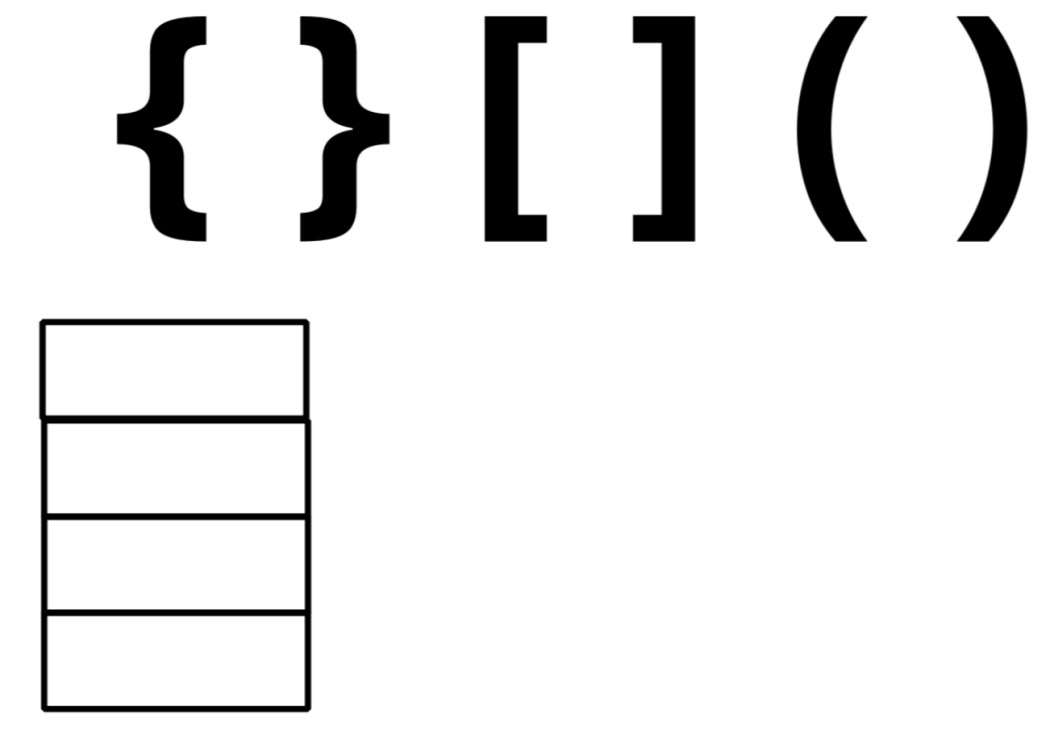
Entrada	Salida
`exp = {}[]()`	`True`
`exp = {[[}[]](])`	`False`
`exp = [(()](()[)`	`False`
`exp = {{}}()[[]]`	`True`
`exp = {{{`	`False`

Solución

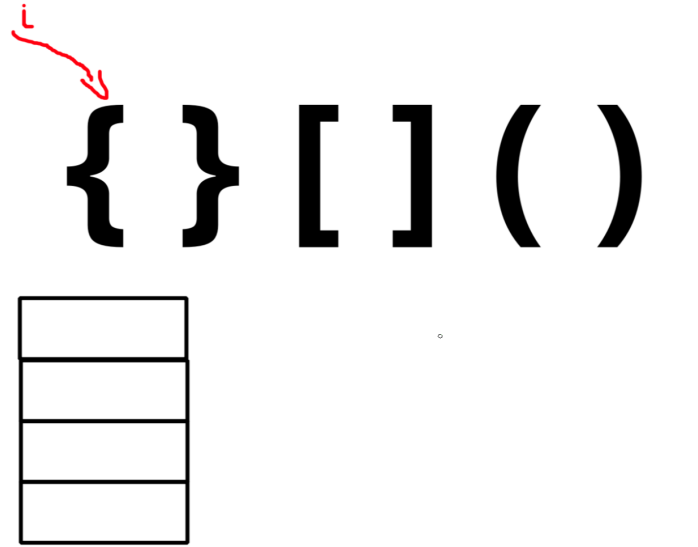
Libro “Data Structures and Algorithm Analysis”[1] capítulo 3 “Lists, Stacks and Queues” sección 3.6.3 “Applications”.

> Make an empty stack. Read characters until end of file. If the character is an opening symbol, push it onto the stack. If it is a closing symbol and the stack is empty, report an error. Otherwise, pop the stack. If the symbol popped is not the corresponding opening symbol, then report an error. At end of file, if the stack is not empty, report an error.

Pasos

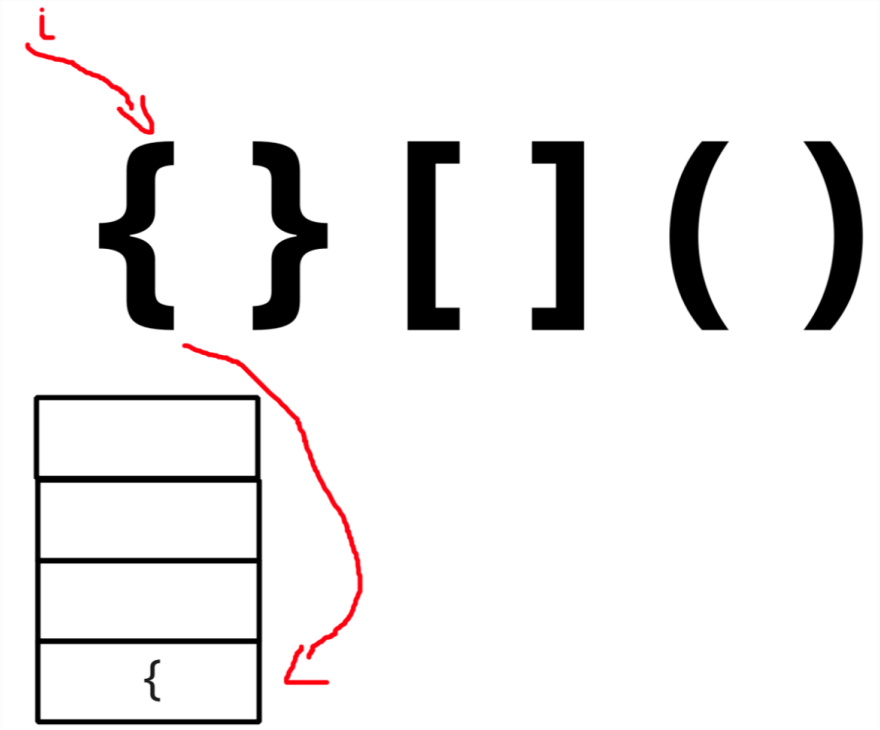


Pasos

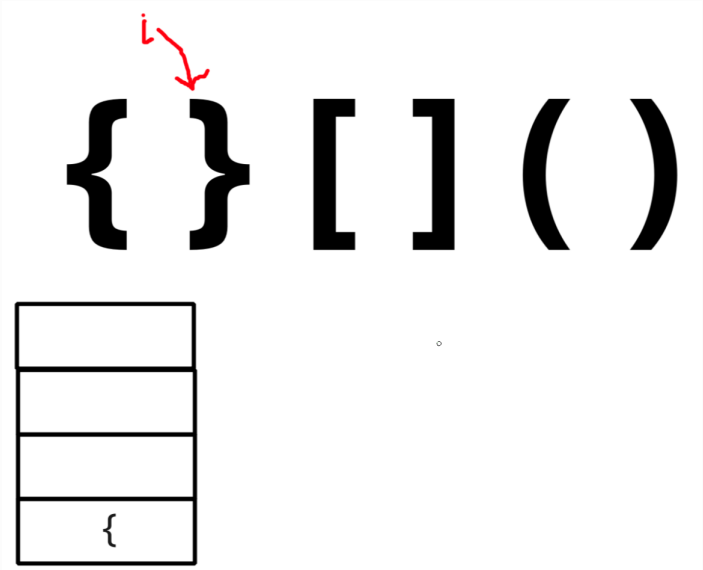


- ¿Es *i* un símbolo abierto?
- Sí, mover al *stack*.

Pasos

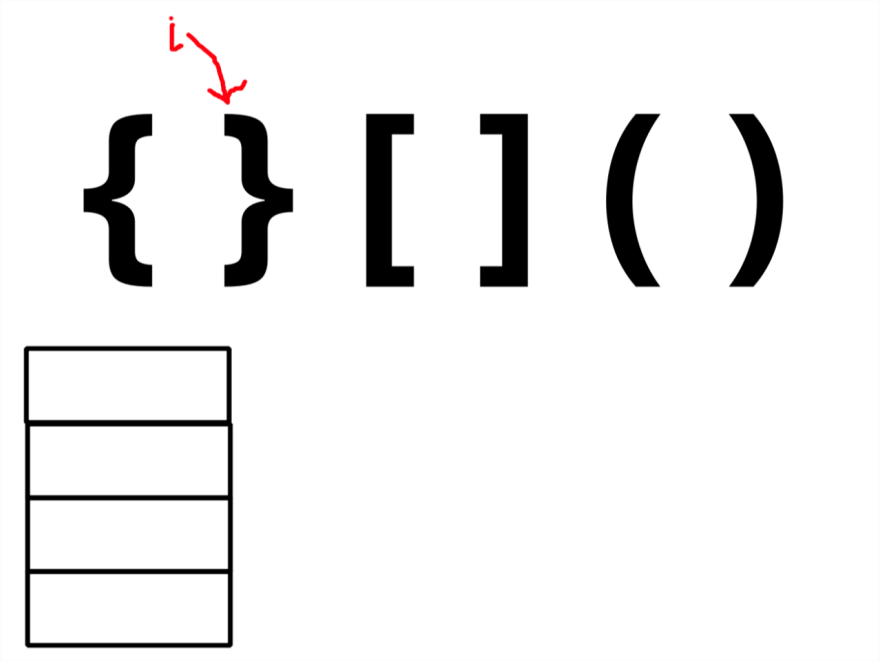


Pasos

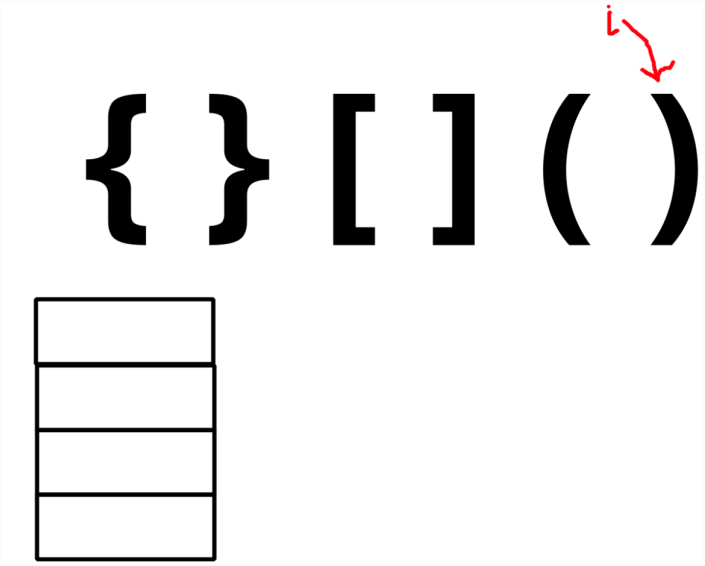


- ¿Es *i* símbolo cerrado?
- Sí, ¿Está vacía la cola?
- No, hacer **pop** al *stack*

Pasos



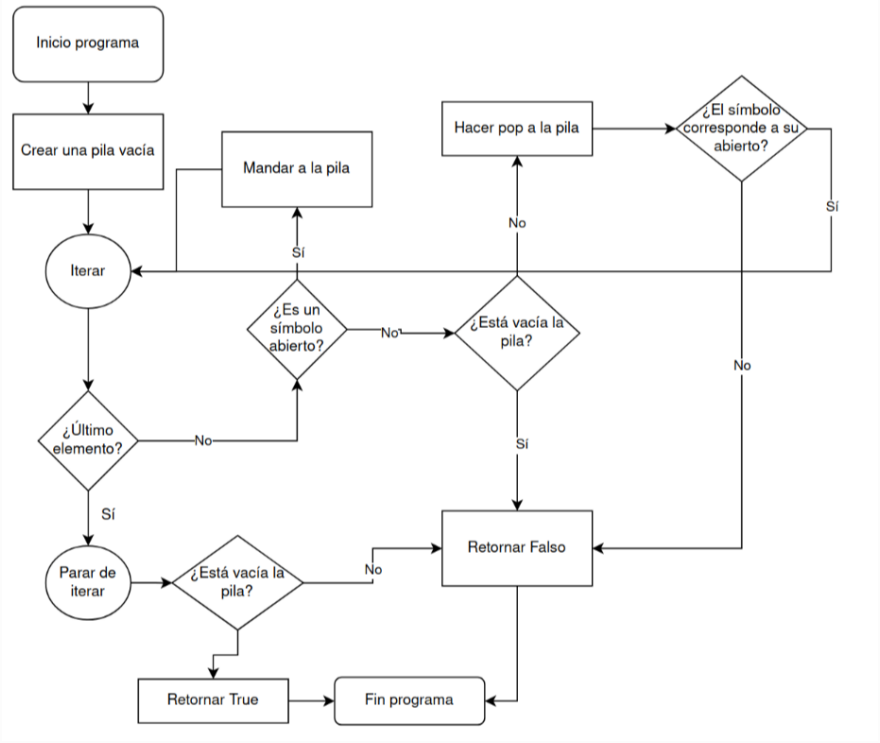
Pasos



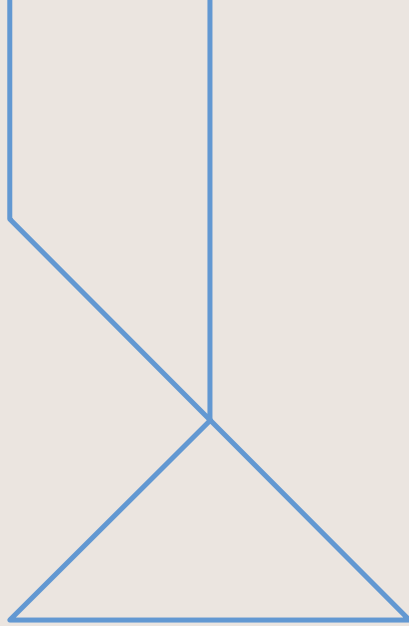
- ¿Está el *stack* vacío?
- Sí, devolver `True`.

Diagrama de flujo

Diagrama de flujo



UTADEO



Implementación En C++

Implementación en C++

```cpp
#include <bits/stdc++.h>

#define endl "\n"

using namespace std;


bool check(){
    string exp {};
    stack<char> s {};
    cin >> exp;
    auto n {size(exp)};
    cout << boolalpha;
    for(int i = 0; i < n; i++){
        if (exp[i] == '{' || exp[i] == '[' || exp[i] == '(' )
            s.push(exp[i]);
        if (exp[i] == '}' && s.top() == '{')
            s.pop();
        if (exp[i] == ']' && s.top() == '[')
            s.pop();
        if (exp[i] == '(' && s.top() == '(')
            s.pop();
    }
    if (s.size() == 0)
        return true;
    return false;
}



int main(){

    cout << check() << endl;

    return 0;
}
```

UTADEO

Complejidad temporal y espacial

Complejidad temporal y espacial

- Temporal:

$$O(n)$$

- Espacial:

$$O(n)$$

Referencias

Referencias I

[1] Mark Allen Weiss. *Data Structures and Algorithm Analysis*. Retrieved from https://www.uoitc.edu.iq/images/documents/informatics-institute/Competitive_exam/DataStructures.pdf. Pearson, 2020.