



Ejercicios del profe: Pilas & Colas 2025-1

Ludwig Alvarado Becerra

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

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

- $1 \leq \text{exp.length} \leq 10^3$
- La cadena solo contendrá los siguientes caracteres: (,), [,], { y }

Problema 1 | Tests

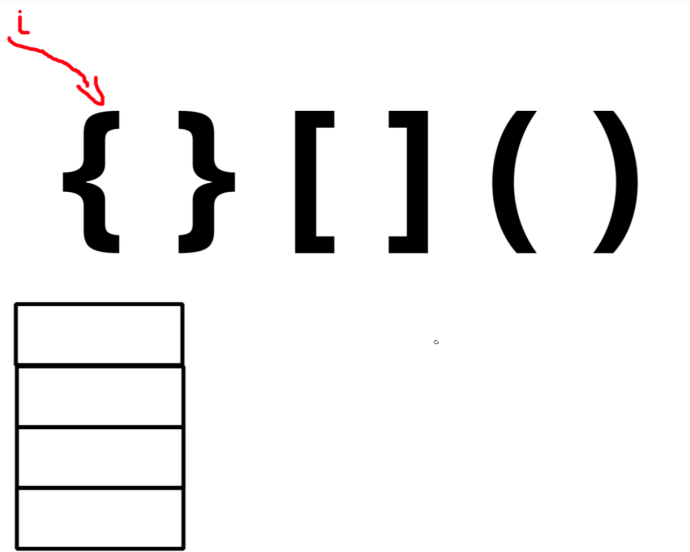
Entrada	Salida
exp = {}[]()	True
exp = {[[]][[]]}()	False
exp = [([])([]	False
exp = {[]}{}()[[]]	True
exp = {{{	False

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

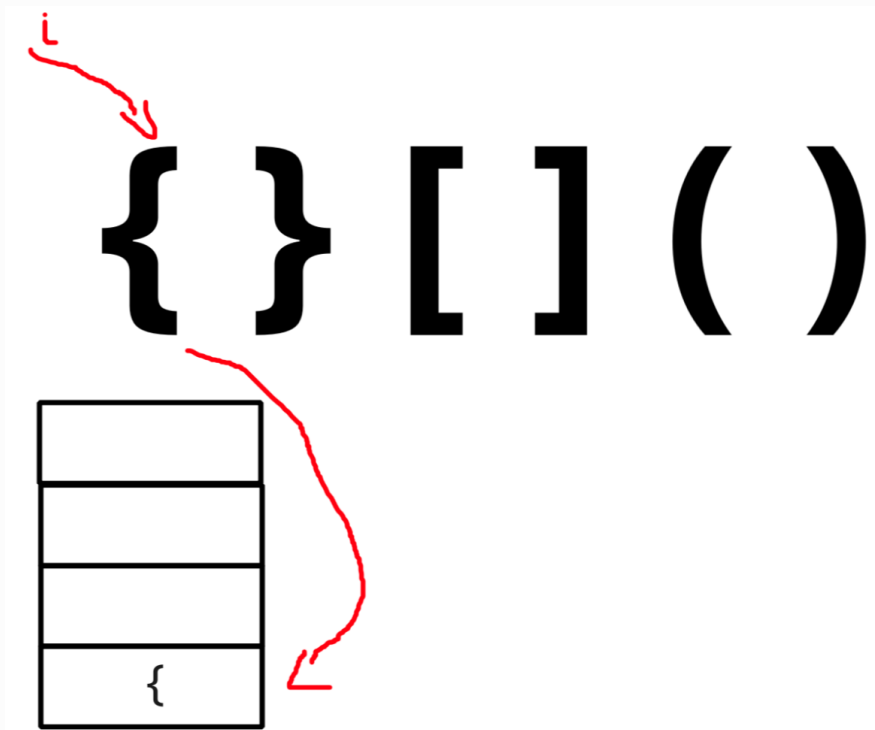
Make an empty stack. Read characters until end of file. If the character is an opening symbol, push it onto the stack. If it is a closing symbol and the stack is empty, report an error. Otherwise, pop the stack. If the symbol popped is not the corresponding opening symbol, then report an error. At end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

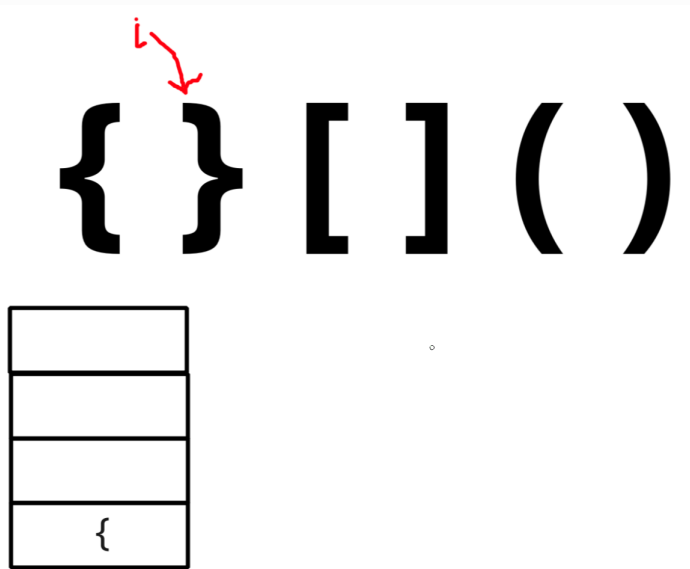
{ } [] ()



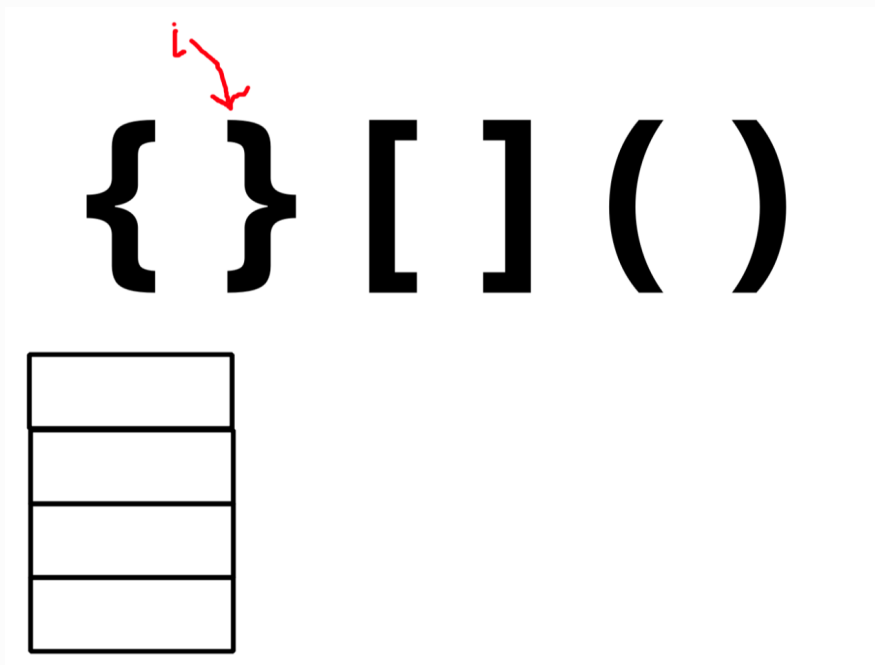


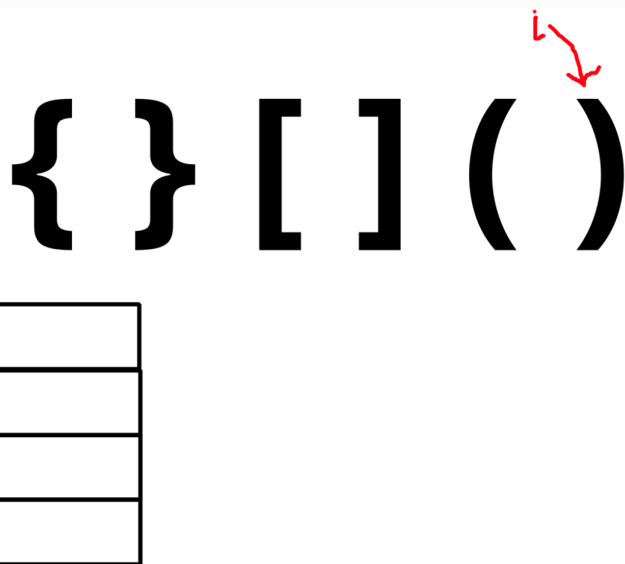
- ¿Es *i* un símbolo abierto?
- Sí, mover al *stack*.





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

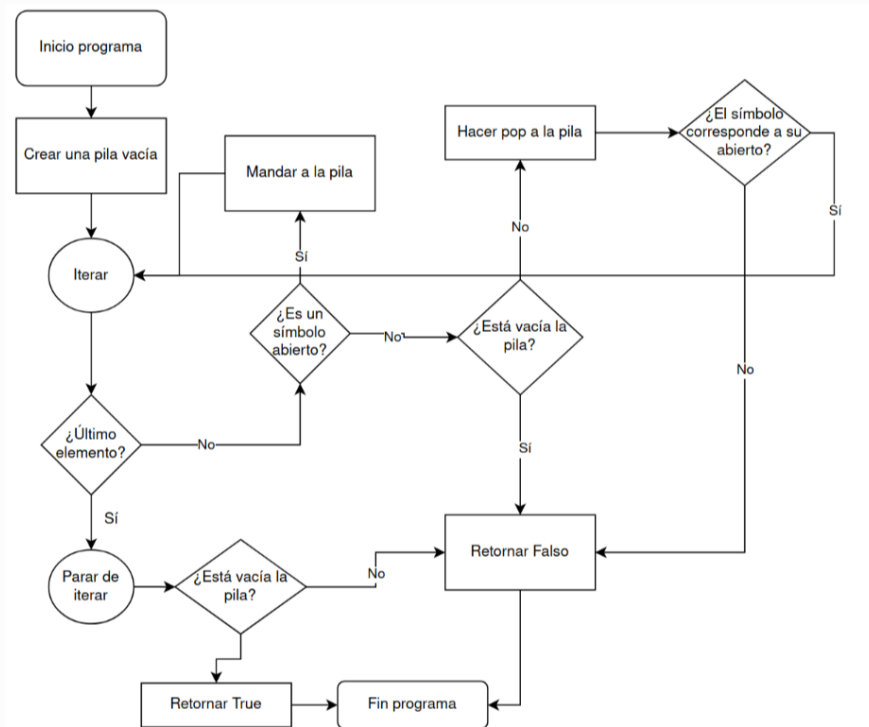




- ¿Está el *stack* vacío?
- Sí, devolver True.

Diagrama de flujo

Diagrama de flujo



Implementación En C++

Implementación en C++

```

1 #include <bits/stdc++.h>
2
3 #define endl "\n"
4
5 using namespace std;
6
7
8 bool check(){
9     string exp {};
10    stack<char> s {};
11    cin >> exp;
12    auto n {size(exp)};
13    cout << boolalpha;
14    for(int i = 0; i < n; i++){
15        if (exp[i] == '{' || exp[i] == '[' || exp[i] == '(' )
16            s.push(exp[i]);
17        if (exp[i] == '}' && s.top() == '{')
18            s.pop();
19        if (exp[i] == ']' && s.top() == '[')
20            s.pop();
21        if (exp[i] == ')' && s.top() == '(')
22            s.pop();
23    }
24    if (s.size() == 0)
25        return true;
26    return false;
27 }
28
29
30
31 int main(){
32
33     cout << check() << endl;
34
35     return 0;
36 }

```

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.

