



Estructura de Datos

Maria C. Torres

Maria C. Torres

Ing. Electrónica (UNAL)

M.E. Ing. Eléctrica (UPRM)

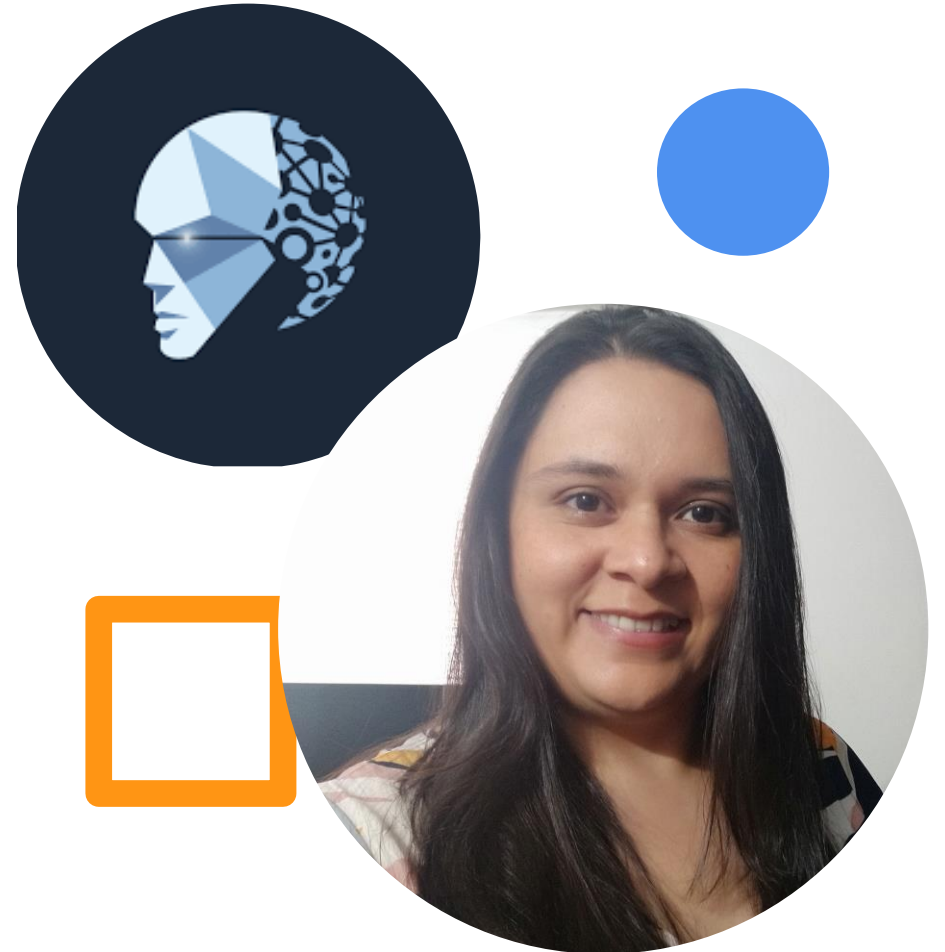
Ph.D. Ciencias e Ingeniería de la Computación y la
Información (UPRM)


Profesora asociada

Dpto. Ciencias de la Computación y la Decisión


mctorresm@unal.edu.co

HORARIO DE ATENCIÓN: Martes 10:00 am a
12:00 m – Oficina 313 M8A





Contenido del Curso

- 
- ☐ Introducción: revisión fundamentos y POO
 - ☐ Análisis de complejidad
 - ☐ Arreglos
 - ☐ Listas enlazadas
 - ☐ **Pilas y colas**
 - ☐ Heap
 - ☐ Árboles binarios
 - ☐ Tablas hash
 - ☐ Grafos



Pilas y colas

- ❑ Pilas – Stacks
 - ❑ Implementación usando arreglos y lista simple
- ❑ Colas – Queue
 - ❑ Implementación usando arreglos y lista simple
- ❑ **Análisis de complejidad operaciones**
- ❑ Aplicaciones y algoritmos

ANÁLISIS DE ALGORITMOS

ArrayStack	Stack
- data[]: Object - top: int	- data: List
+ ArrayStack(int capacity) + size(): int + isEmpty(): Boolean + push(Object e) + pop(): Object + top(): Object	+ Stack() + size(): int + isEmpty(): Boolean + push(Object e) + pop(): Object + top(): Object

Operación	Complejidad
push()	$\Theta(1)$
pop()	$\Theta(1)$
top()	$\Theta(1)$

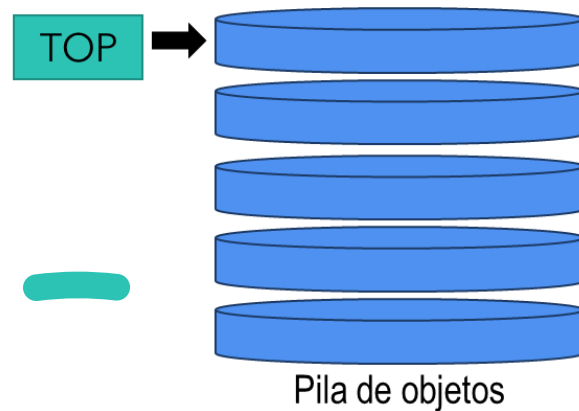
ArrayQueue	Queue
- data[]: Object - first: int - rear: int	- data: List
+ ArrayQueue(int capacity) + size(): int + isEmpty(): boolean + enqueue(Object e) + dequeue(): Object e + first(): Object e	+ Queue() + size(): int + isEmpty(): Boolean + enqueue(Object e) + dequeue(): Object + first(): Object

Operación	Complejidad
enqueue()	$\Theta(1)$
dequeue()	$\Theta(1)$
first()	$\Theta(1)$

USO DE PILAS Y COLAS

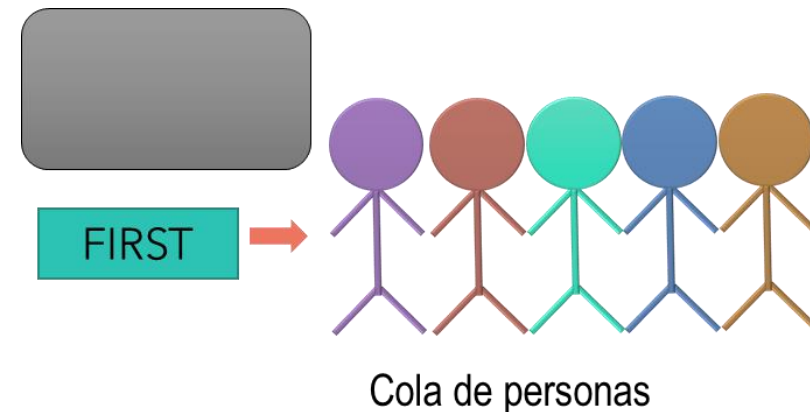
Pilas:

- ❑ Principio LIFO (*Last-to-In First-to-Out*)
- ❑ Arreglos: memoria fija - tamaño limitado
- ❑ Listas: memoria dinámica - sin capacidad limitada



Colas:

- ❑ Principio FIFO (*First-to-In First-to-Out*)
- ❑ Arreglos: memoria fija - tamaño limitado
- ❑ Listas: memoria dinámica - sin capacidad limitada



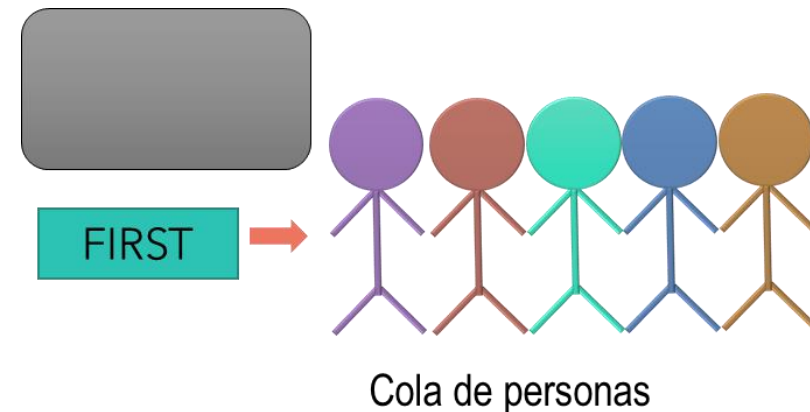
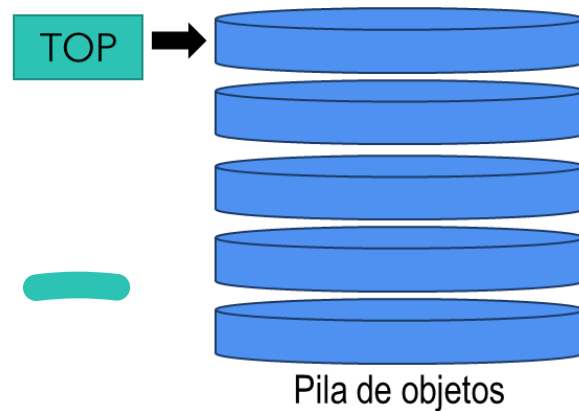


Pilas y colas

- ❑ Pilas – Stacks
 - ❑ Implementación usando arreglos y lista simple
- ❑ Colas – Queue
 - ❑ Implementación usando arreglos y lista simple
- ❑ Análisis de complejidad operaciones
- ❑ **Aplicaciones y algoritmos**

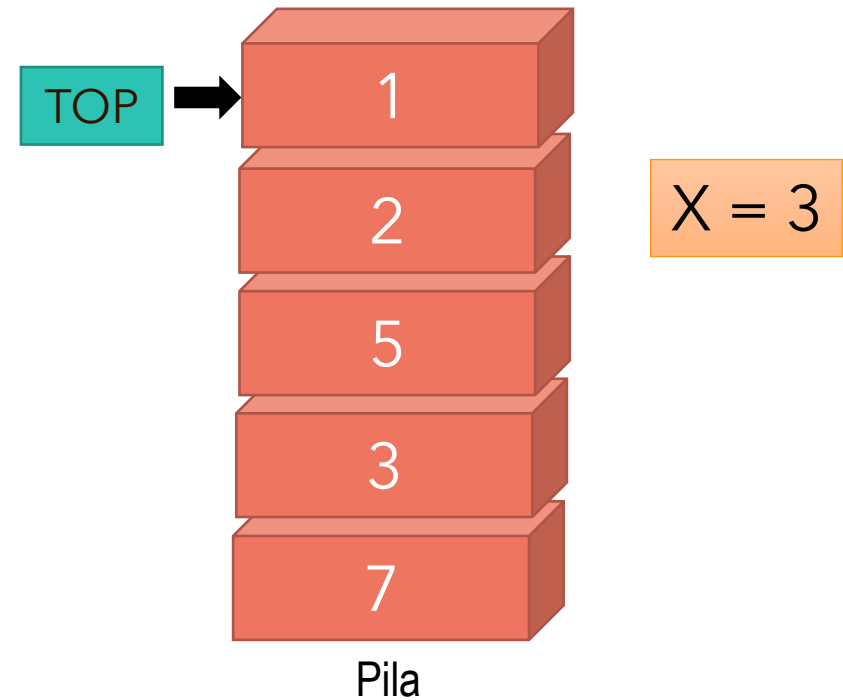
USO DE PILAS Y COLAS

Problema 1: Presente el pseudocódigo para un método que recibe una pila S con n números enteros y un valor x . El método debe buscar el valor x dentro de la pila, empleando como variable auxiliar una cola Q . Si el valor x está en la pila S debe retornar verdadero, en caso contrario falso. La pila S debe quedar con todos los números enteros en el orden original



USO DE PILAS Y COLAS

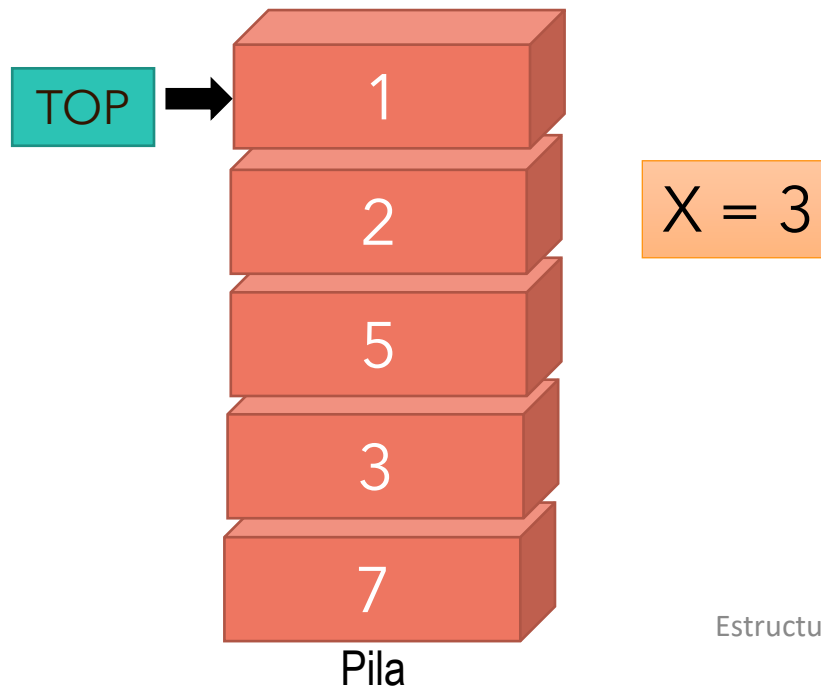
Problema 1: Presente el pseudocódigo para un método que recibe una pila S con n números enteros y un valor x . El método debe buscar el valor x dentro de la pila, empleando como variable auxiliar una cola Q . Si el valor x está en la pila S debe retornar verdadero, en caso contrario falso. La pila S debe quedar con todos los números enteros en el orden original



USO DE PILAS Y COLAS

Algoritmo:

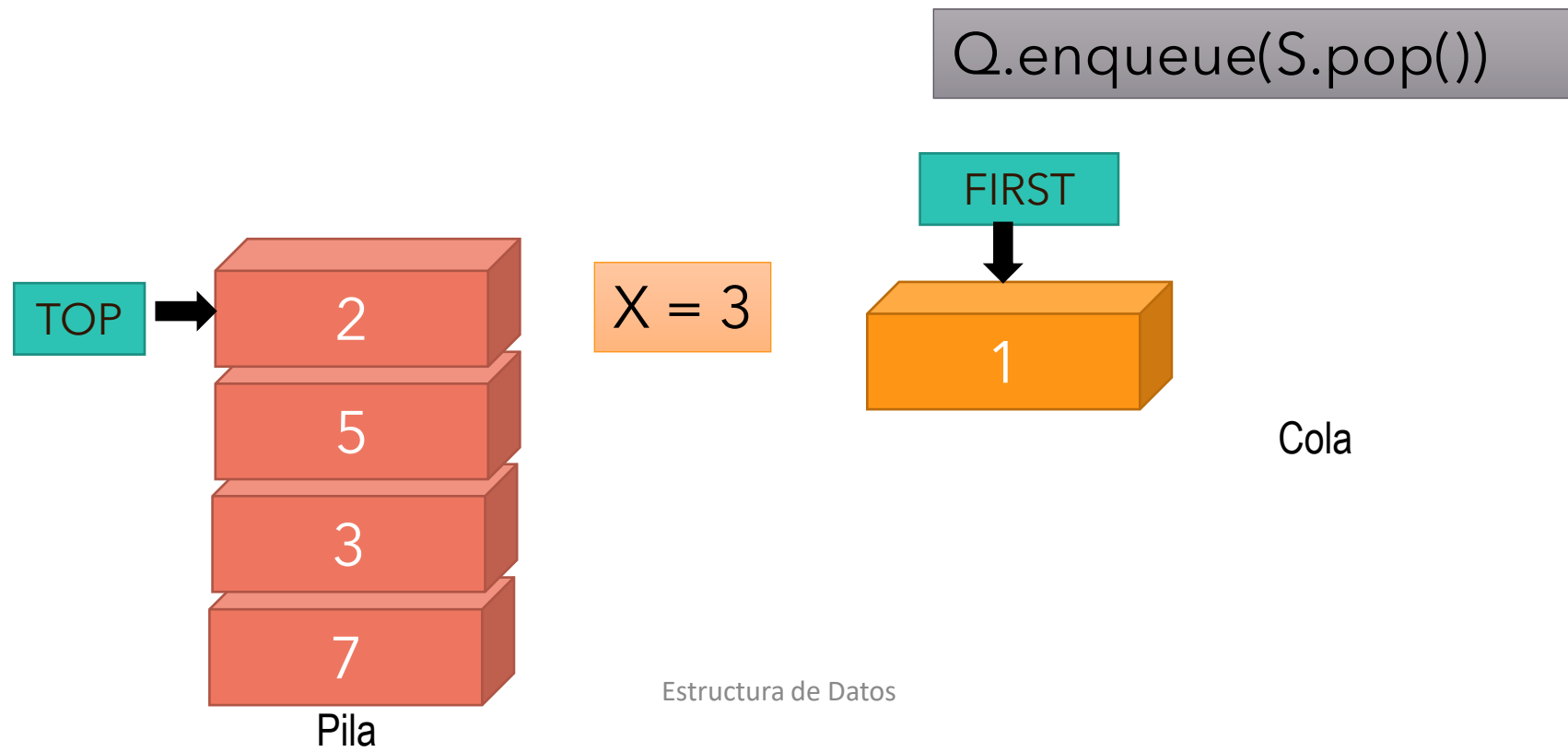
1. Sacamos cada elemento de la pila y lo comparamos con X
 - a. Cada elemento que vamos sacando lo almacenamos temporalmente en una cola
 - b. Si encontramos X, devolvemos los elementos de la cola a la pila



USO DE PILAS Y COLAS

Algoritmo:

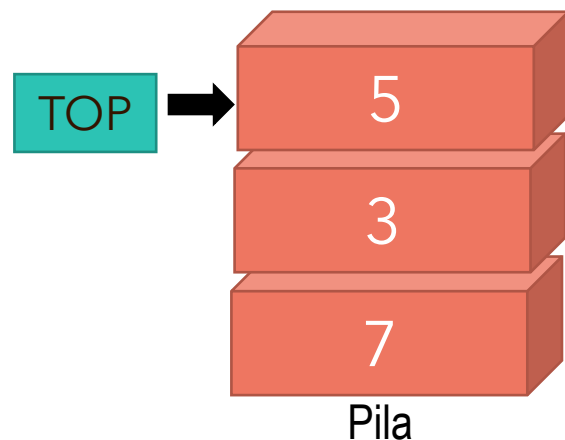
1. Sacamos cada elemento de la pila y lo comparamos con X
 - a. Cada elemento que vamos sacando lo almacenamos temporalmente en una cola
 - b. Si encontramos X, devolvemos los elementos de la cola a la pila



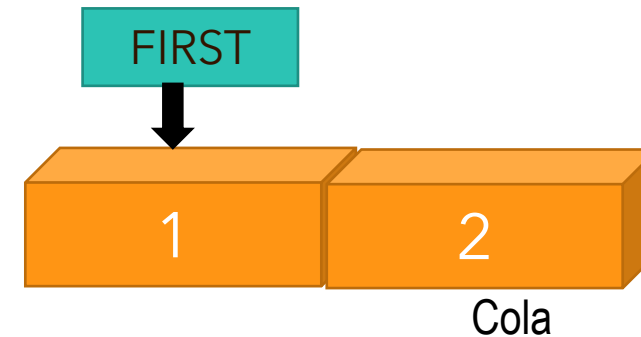
USO DE PILAS Y COLAS

Algoritmo:

1. Sacamos cada elemento de la pila y lo comparamos con X
 - a. Cada elemento que vamos sacando lo almacenamos temporalmente en una cola
 - b. Si encontramos X, devolvemos los elementos de la cola a la pila



X = 3



```
Q.enqueue(S.pop())
```

USO DE PILAS Y COLAS

Algoritmo:

1. Sacamos cada elemento de la pila y lo comparamos con X
 - a. Cada elemento que vamos sacando lo almacenamos temporalmente en una cola
 - b. Si encontramos X, devolvemos los elementos de la cola a la pila**

```
Q.enqueue(S.pop())
```

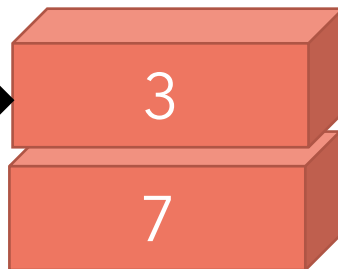
FIRST

X = 3



Cola

TOP



Pila

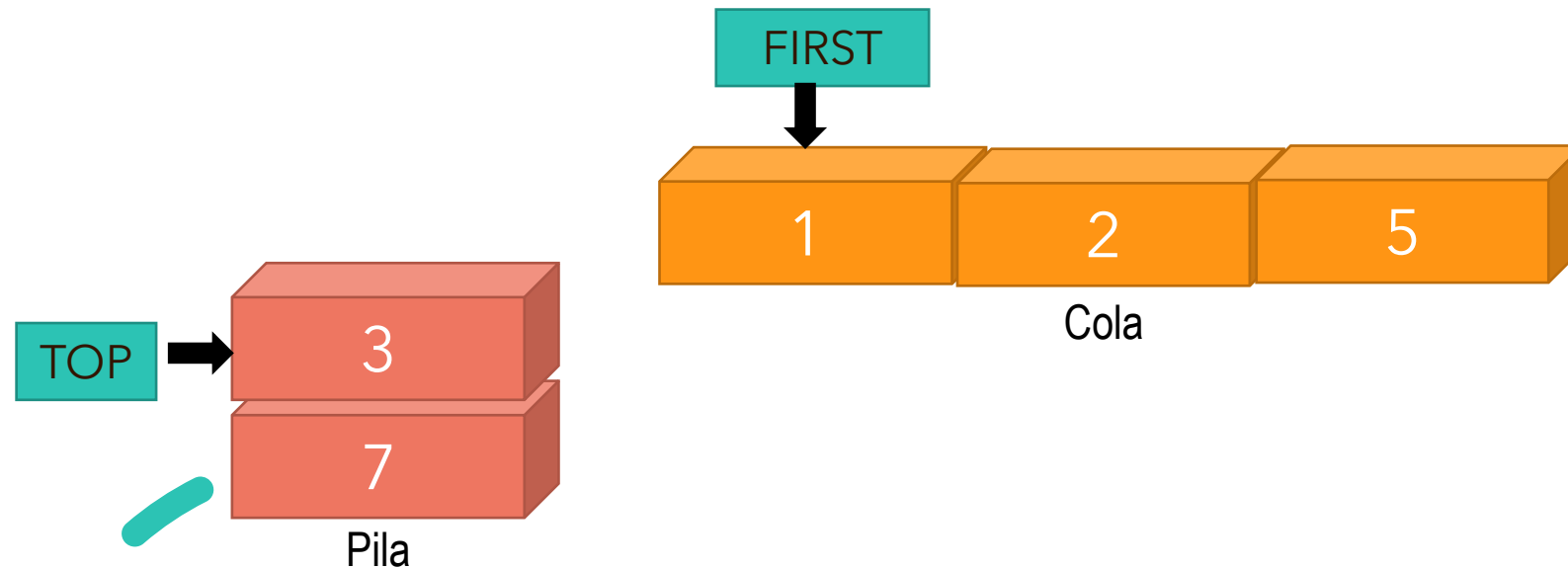
Valor encontrado!!!

USO DE PILAS Y COLAS

Algoritmo:

1. Sacamos cada elemento de la pila y lo comparamos con X
 - a. Cada elemento que vamos sacando lo almacenamos temporalmente en una cola
 - b. Si encontramos X, devolvemos los elementos de la cola a la pila**

```
temp.push(Q.dequeue())
```

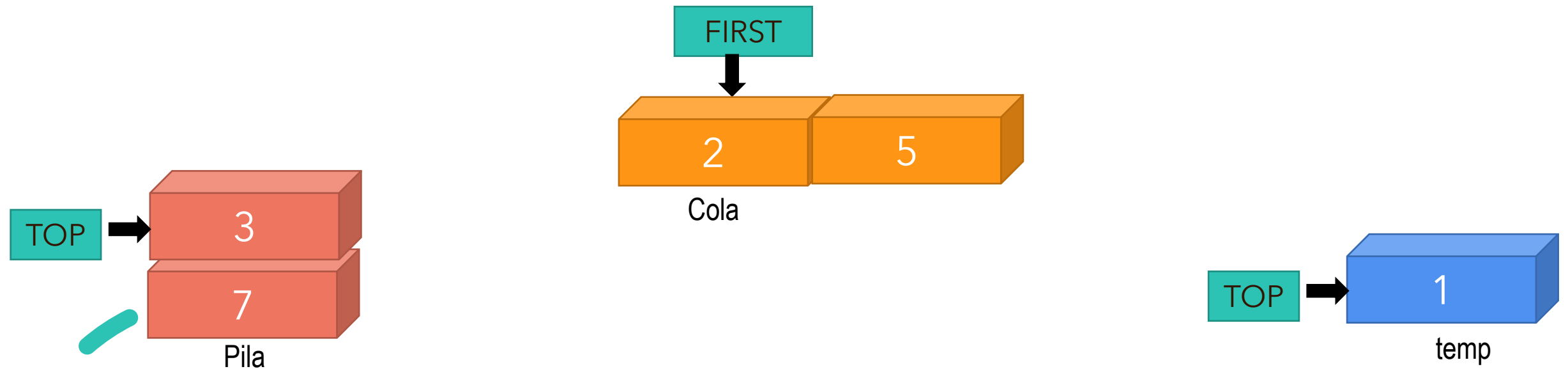


USO DE PILAS Y COLAS

Algoritmo:

1. Sacamos cada elemento de la pila y lo comparamos con X
 - a. Cada elemento que vamos sacando lo almacenamos temporalmente en una cola
 - b. Si encontramos X, devolvemos los elementos de la cola a la pila**

```
temp.push(Q.dequeue())
```

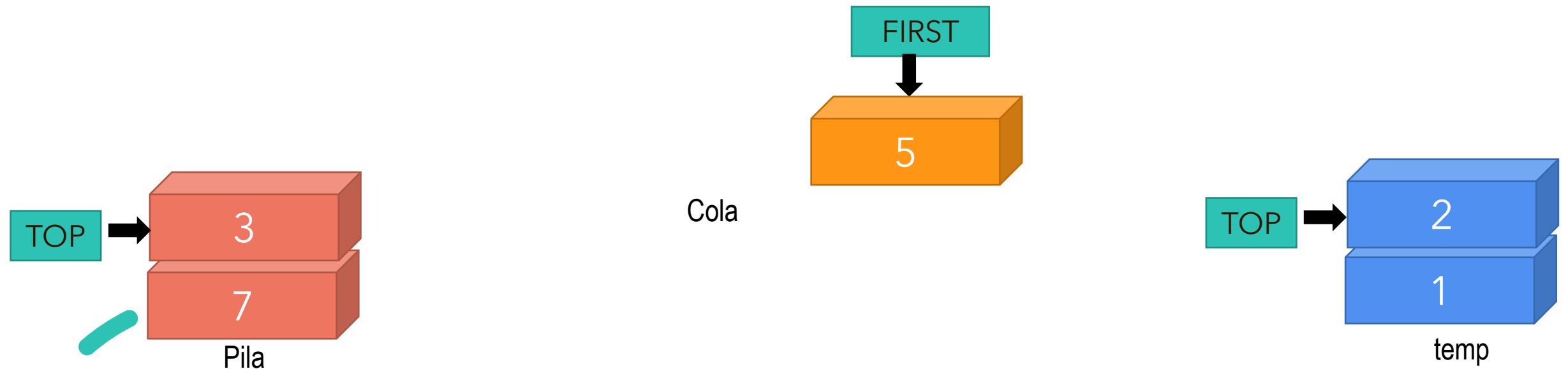


USO DE PILAS Y COLAS

Algoritmo:

1. Sacamos cada elemento de la pila y lo comparamos con X
 - a. Cada elemento que vamos sacando lo almacenamos temporalmente en una cola
 - b. Si encontramos X, devolvemos los elementos de la cola a la pila**

```
temp.push(Q.dequeue())
```

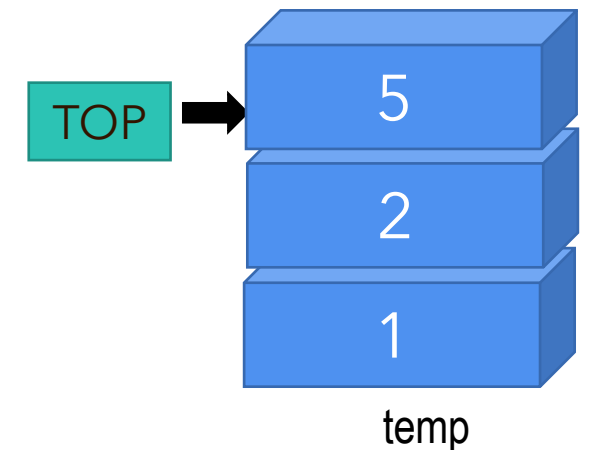
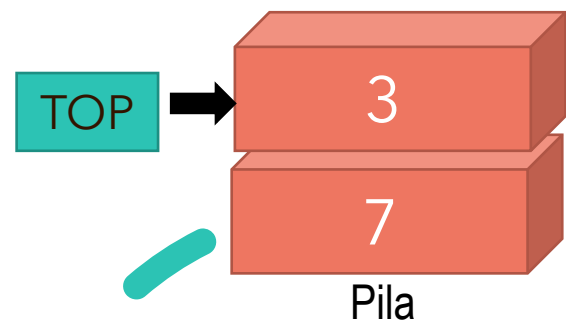


USO DE PILAS Y COLAS

Algoritmo:

1. Sacamos cada elemento de la pila y lo comparamos con X
 - a. Cada elemento que vamos sacando lo almacenamos temporalmente en una cola
 - b. Si encontramos X, devolvemos los elementos de la cola a la pila**

```
temp.push(Q.dequeue())
```



USO DE PILAS Y COLAS

Algoritmo:

1. Sacamos cada elemento de la pila y lo comparamos con X
 - a. Cada elemento que vamos sacando lo almacenamos temporalmente en una cola
 - b. Si encontramos X, devolvemos los elementos de la cola a la pila**

```
S.push(temp.pop())
```

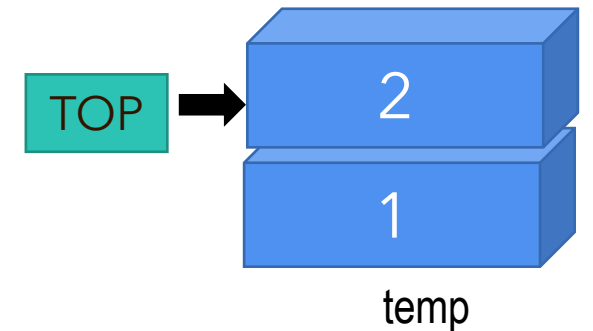
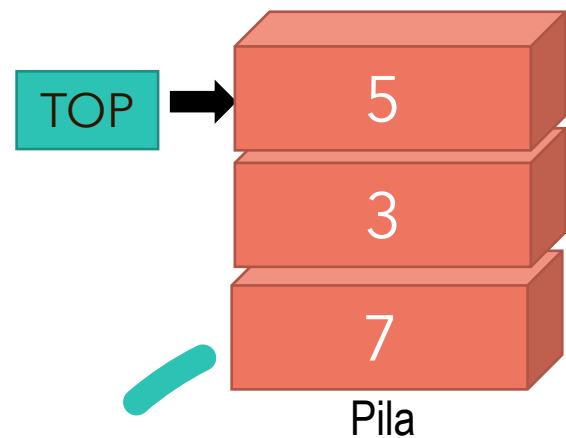


USO DE PILAS Y COLAS

Algoritmo:

1. Sacamos cada elemento de la pila y lo comparamos con X
 - a. Cada elemento que vamos sacando lo almacenamos temporalmente en una cola
 - b. Si encontramos X, devolvemos los elementos de la cola a la pila**

```
S.push(temp.pop())
```

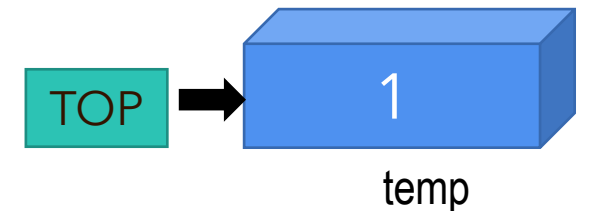
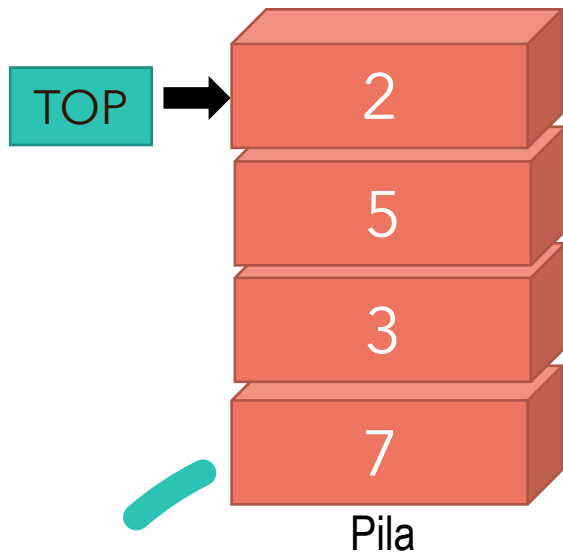


USO DE PILAS Y COLAS

Algoritmo:

1. Sacamos cada elemento de la pila y lo comparamos con X
 - a. Cada elemento que vamos sacando lo almacenamos temporalmente en una cola
 - b. Si encontramos X, devolvemos los elementos de la cola a la pila**

```
S.push(temp.pop())
```

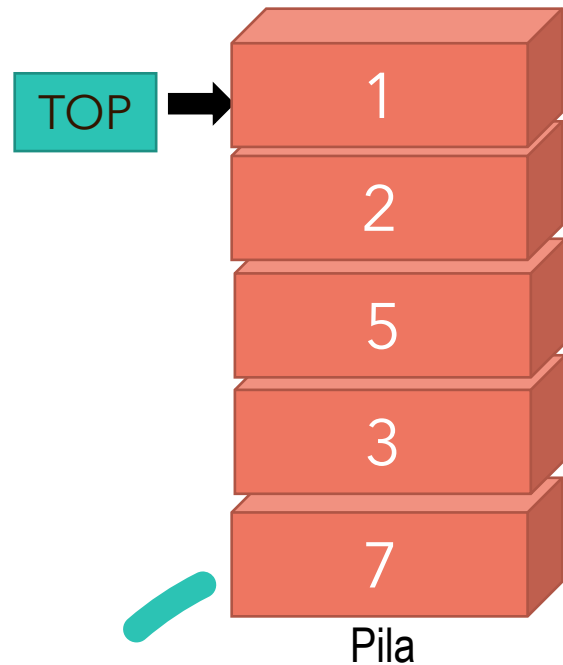


USO DE PILAS Y COLAS

Algoritmo:

1. Sacamos cada elemento de la pila y lo comparamos con X
 - a. Cada elemento que vamos sacando lo almacenamos temporalmente en una cola
 - b. Si encontramos X, devolvemos los elementos de la cola a la pila**

```
S.push(temp.pop())
```



temp

USO DE PILAS Y COLAS

Algoritmo:

1. Sacamos cada elemento de la pila y lo comparamos con X
 - a. Cada elemento que vamos sacando lo almacenamos temporalmente en una cola
 - b. Si encontramos X, devolvemos los elementos de la cola a la pila**

```
buscar(Stack S, int x)
// inicializamos variables auxiliares
Queue Q = new Queue()
Stack temp = new Stack()
Boolean flag = false
// buscamos x en la pila S
while (!S.isEmpty() && S.top() != x)
    Q.enqueue(S.pop())
if !S.isEmpty()
    //Dato encontrado
    flag = true
//pasamos los datos de la cola a temp
while(!Q.isEmpty())
    temp.push(Q.dequeue())
//volvemos los datos a la pila S
while (!temp.isEmpty())
    S.push(temp.pop())
return flag
```

USO DE PILAS Y COLAS

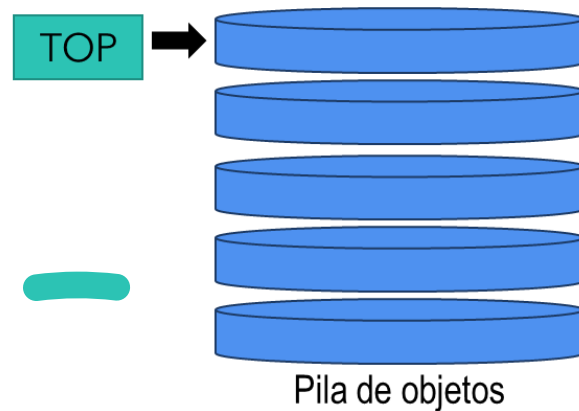
Problema 2: Balanceo de paréntesis

Ahora dirigimos nuestra atención a usar pilas para resolver problemas reales de ciencias de la computación.

En las expresiones aritméticas, por ejemplo:

$$(5-4)*(3-2)/(6-3)$$

los paréntesis deben aparecer de forma balanceada.



Que los paréntesis estén balanceados significa que cada símbolo de apertura tiene un símbolo de cierre correspondiente y que los paréntesis están apropiadamente anidados.

USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

Considere las siguientes cadenas de paréntesis correctamente balanceados:

((()()()()))

(((((()))))

Compare con los siguientes ejemplos que no están balanceadas:

((((((((())

((()()((()

USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

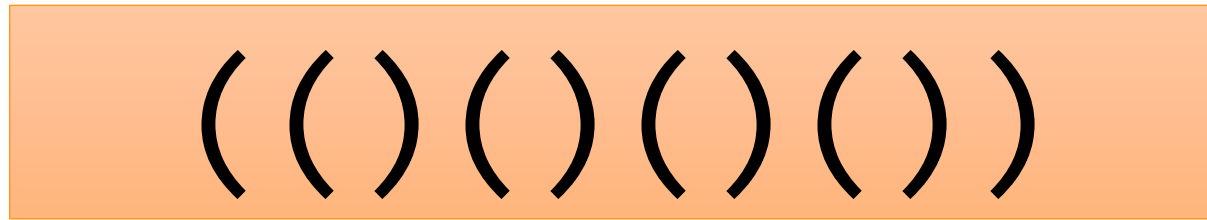
escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.

((()()()()))	✓
(((((()))))	✓
(((((((((((✗
((()()((✗

USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



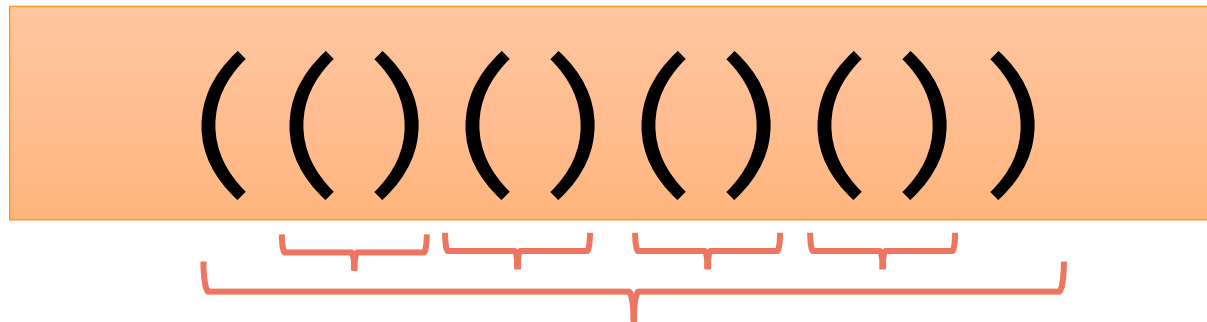
(() () ())

El primer cierre coincide con la
apertura más reciente

USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

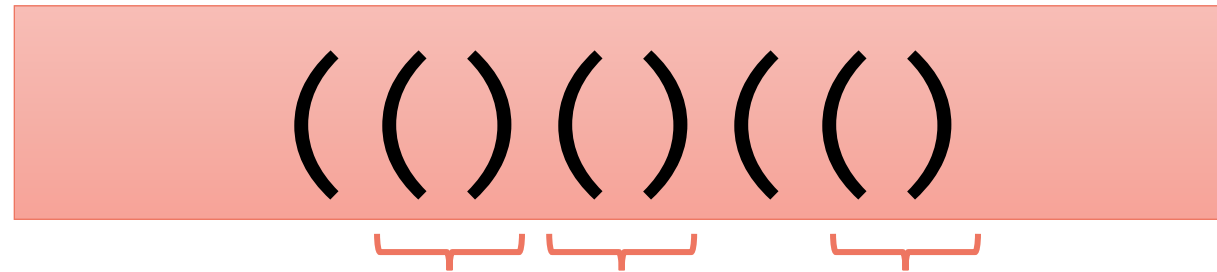
escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



Necesitamos un algoritmo que verifique las correspondencias

Problema 2: Balanceo de paréntesis

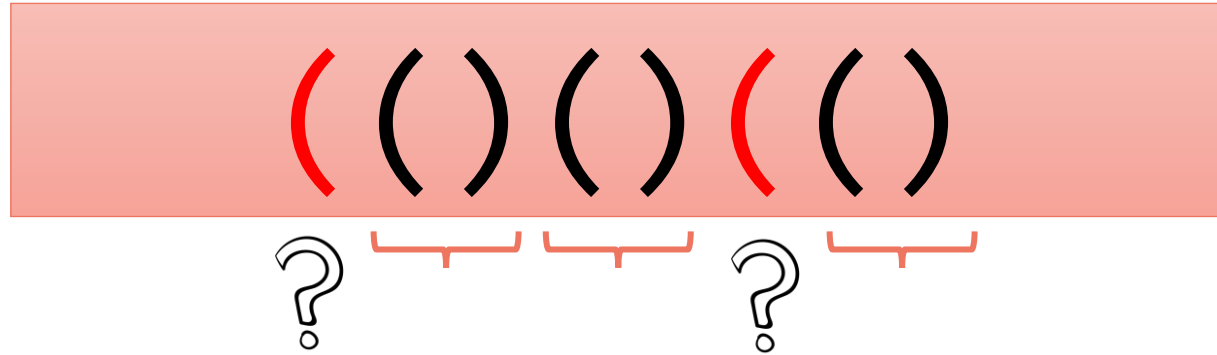
escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



Problema 2: Balanceo de paréntesis

Solución del problema:

- ❑ Vamos a usar un STACK para realizar la verificación de los paréntesis
- ❑ Se debe recorrer la cadena carácter por carácter
 - Si el carácter corresponde a un símbolo de apertura, entonces lo insertamos (PUSH) en el STACK
 - Si el carácter corresponde a un símbolo de cierre, entonces eliminamos (POP) un carácter del STACK
 - Si el STACK está vacío, y no podemos realizar más operaciones POP, entonces la expresión esta desbalanceada
 - Al final, el STACK debe quedar vacío, sino la expresión esta desbalanceada

USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

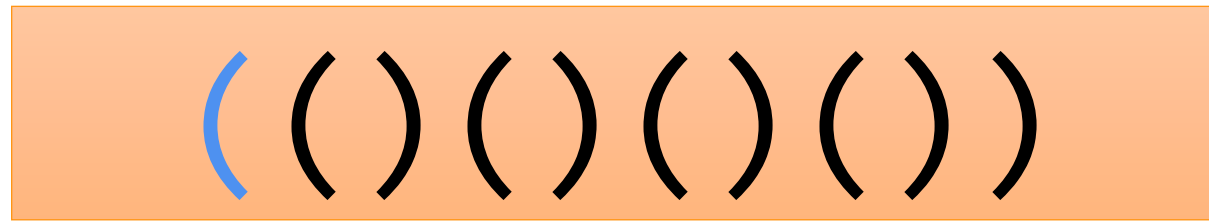
escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.

(() () () ())

USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



push('(')

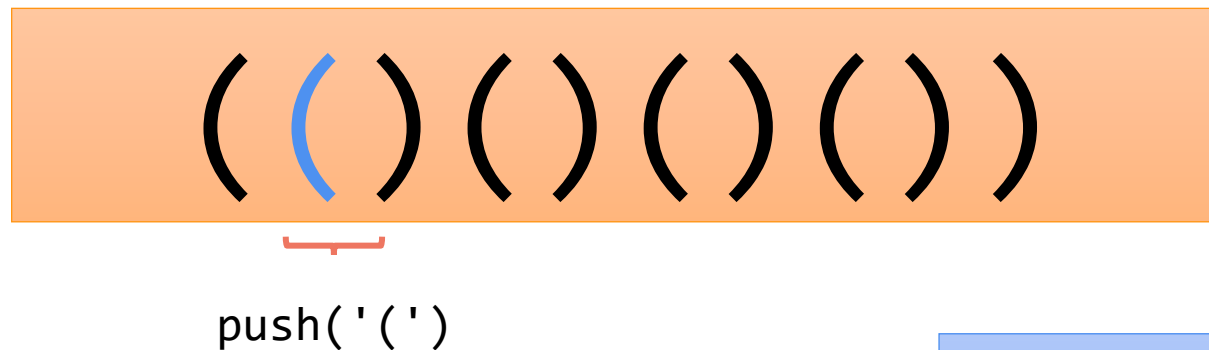
Si el carácter corresponde a un símbolo de apertura, entonces lo insertamos (PUSH) en el STACK



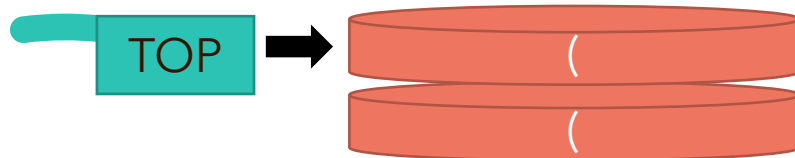
USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



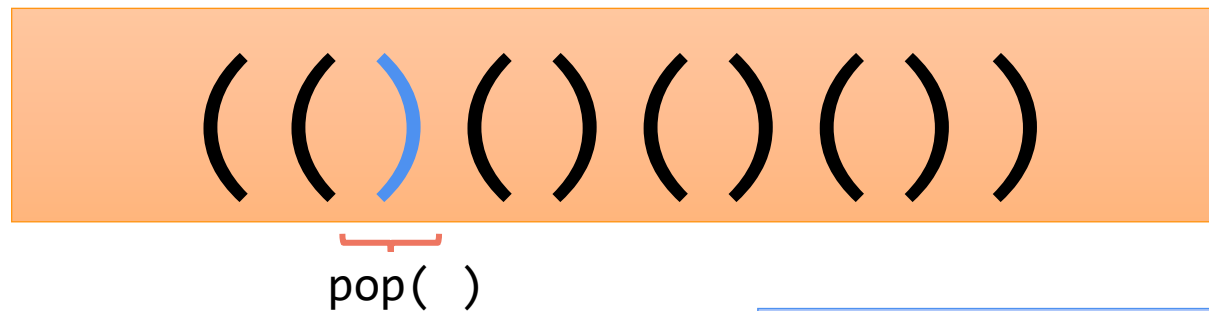
Si el carácter corresponde a un símbolo de apertura, entonces lo insertamos (PUSH) en el STACK



USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



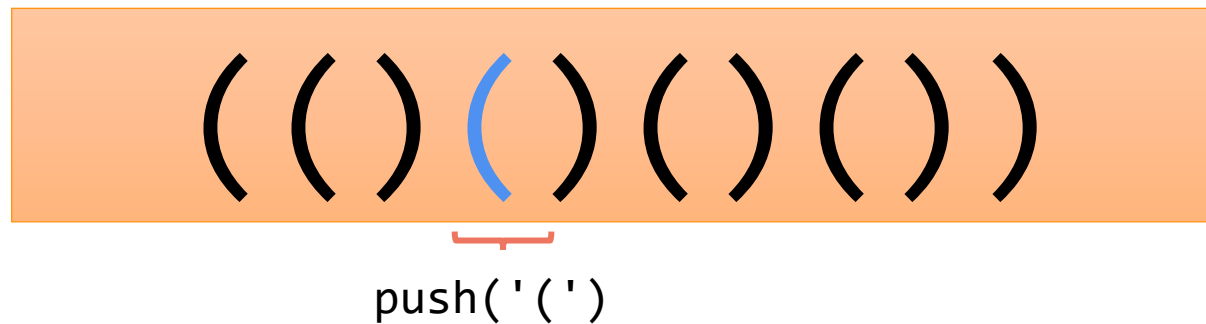
Si el carácter corresponde a un símbolo de cierre, entonces eliminamos (POP) un carácter del STACK



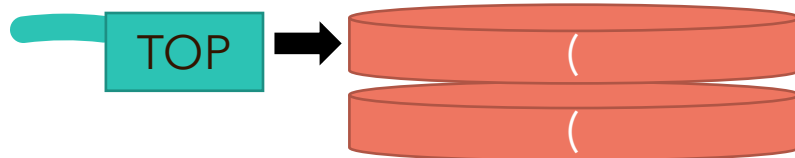
USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



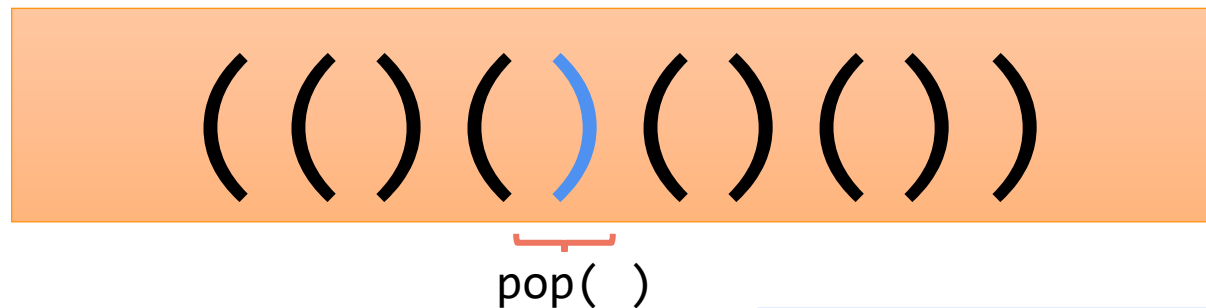
Si el carácter corresponde a un símbolo de apertura, entonces lo insertamos (PUSH) en el STACK



USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



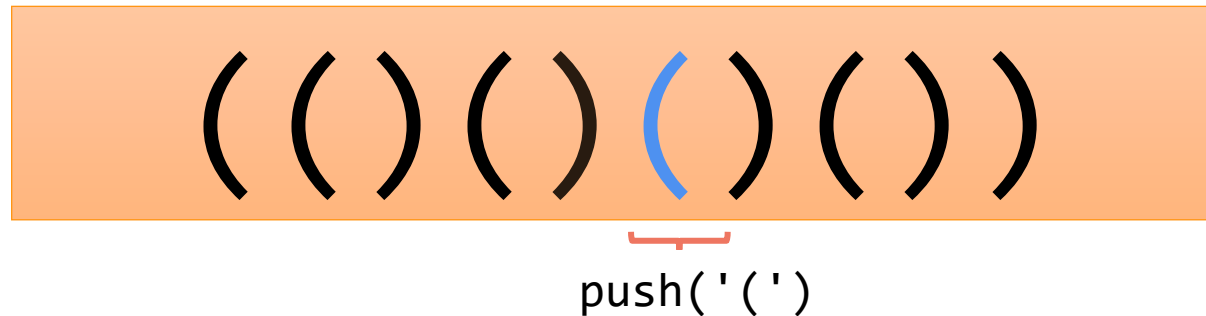
Si el carácter corresponde a un símbolo de cierre, entonces eliminamos (POP) un carácter del STACK



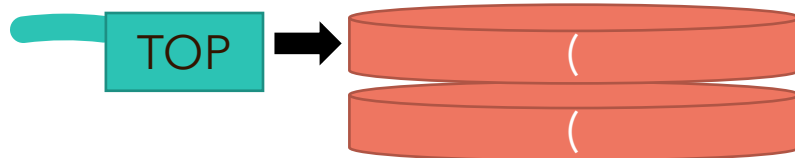
USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



Si el carácter corresponde a un símbolo de apertura, entonces lo insertamos (PUSH) en el STACK



USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.

(() () () ())

pop()

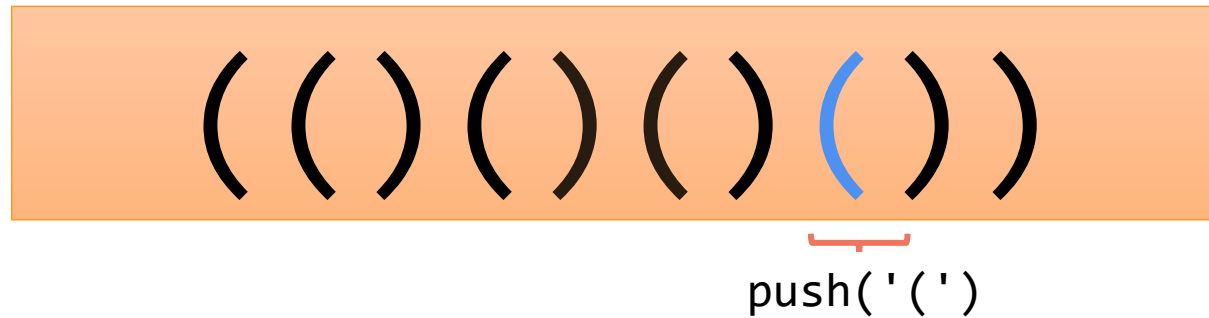
Si el carácter corresponde a un símbolo de cierre, entonces eliminamos (POP) un carácter del STACK



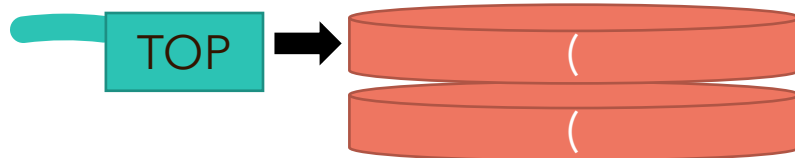
USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



Si el carácter corresponde a un símbolo de apertura, entonces lo insertamos (PUSH) en el STACK



USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.

(() () () ())

pop()

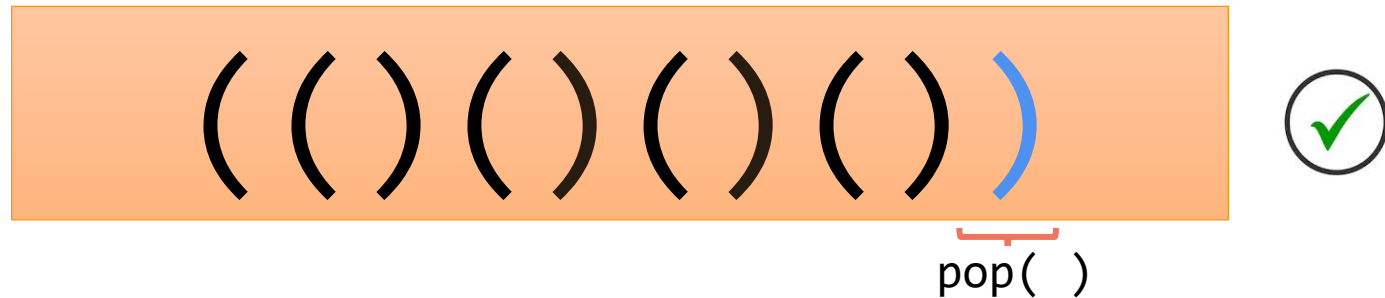
Si el carácter corresponde a un símbolo de cierre, entonces eliminamos (POP) un carácter del STACK



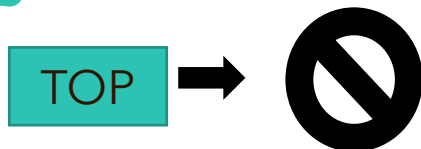
USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



Al final, el STACK debe quedar vacío, sino la expresión esta desbalanceada



USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

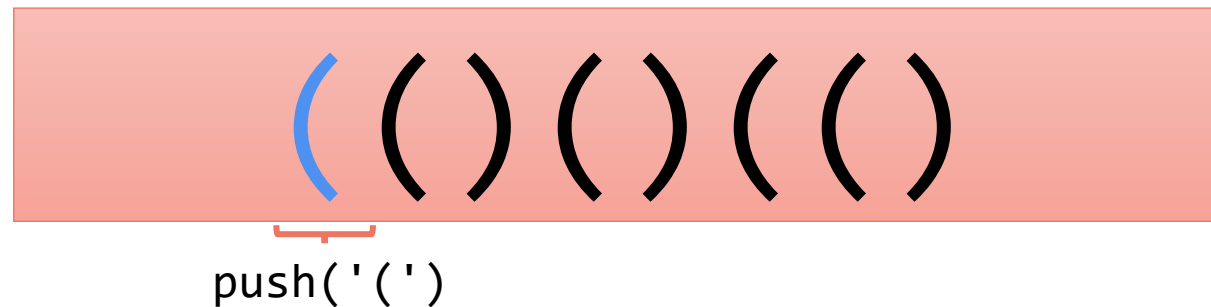
escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.

(() () (()

USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



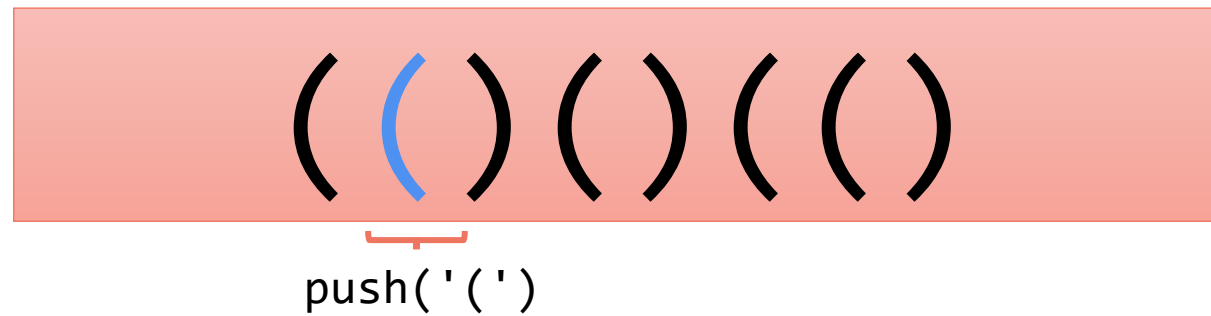
Si el carácter corresponde a un símbolo de apertura, entonces lo insertamos (PUSH) en el STACK



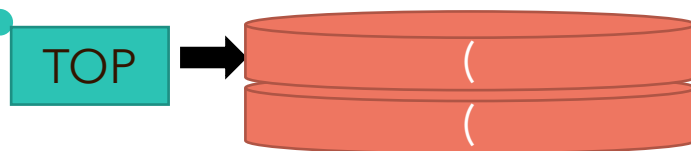
USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



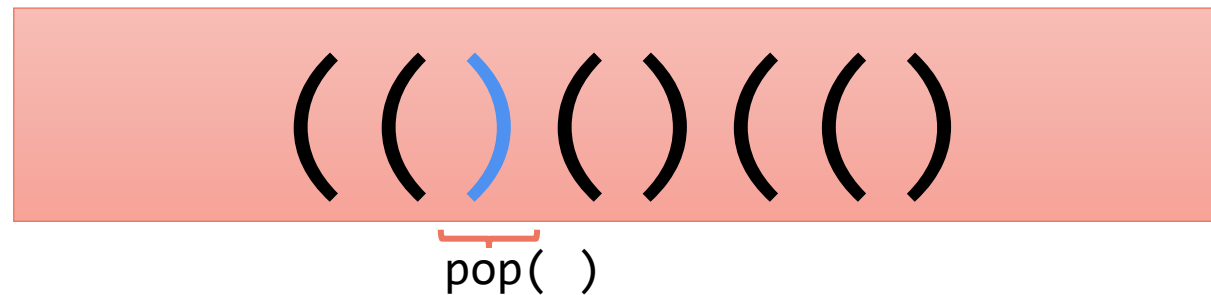
Si el carácter corresponde a un símbolo de apertura, entonces lo insertamos (PUSH) en el STACK



USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



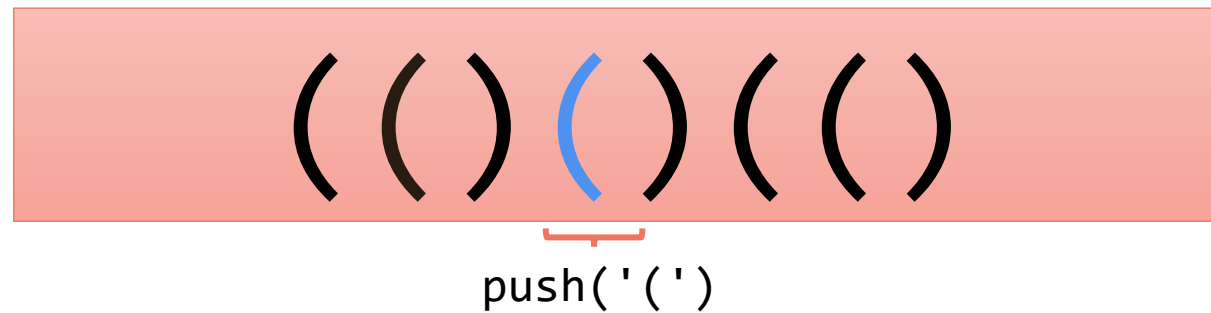
Si el carácter corresponde a un símbolo de cierre, entonces eliminamos (POP) un carácter del STACK



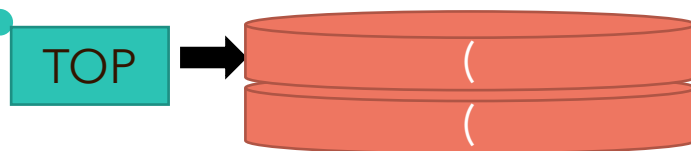
USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



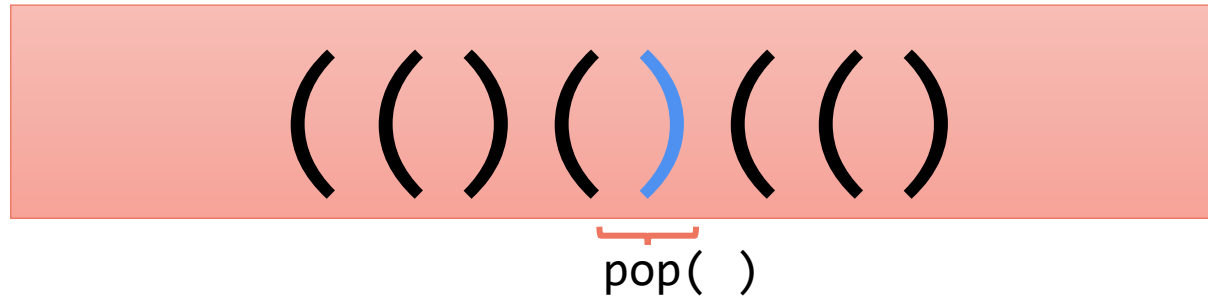
Si el carácter corresponde a un símbolo de apertura, entonces lo insertamos (PUSH) en el STACK



USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



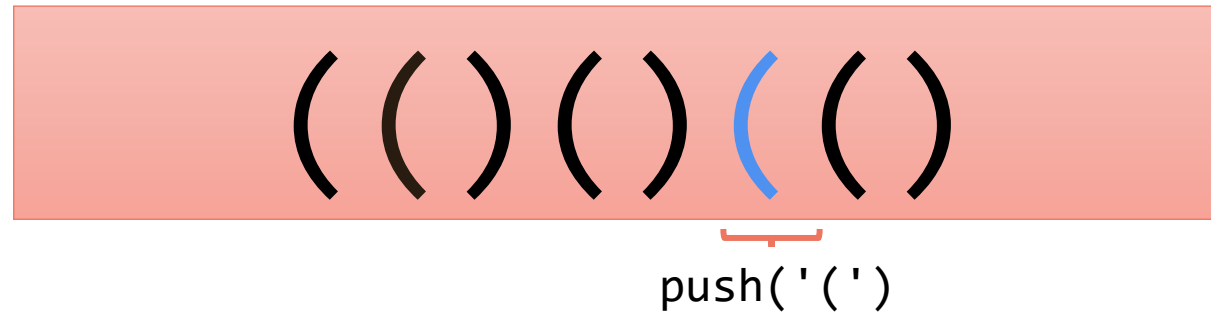
Si el carácter corresponde a un símbolo de cierre, entonces eliminamos (POP) un carácter del STACK



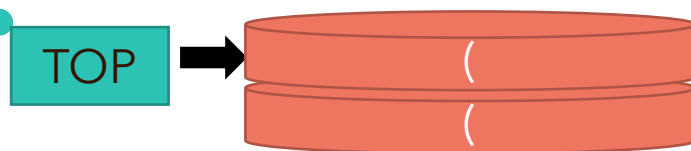
USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



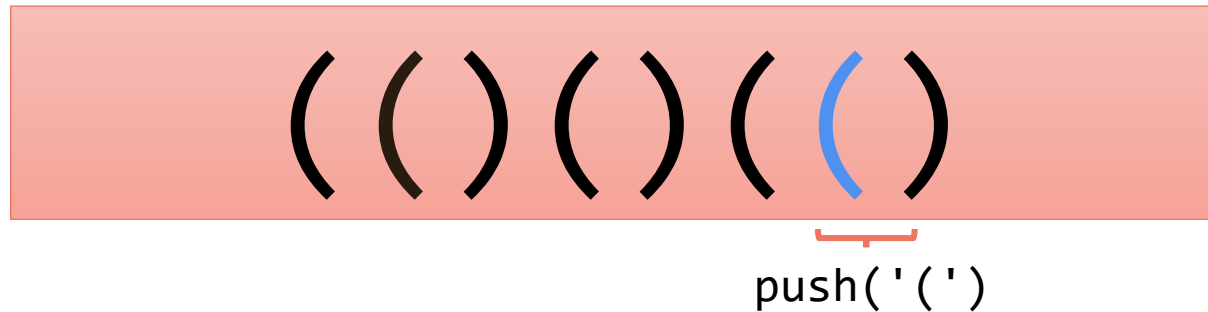
Si el carácter corresponde a un símbolo de apertura, entonces lo insertamos (PUSH) en el STACK



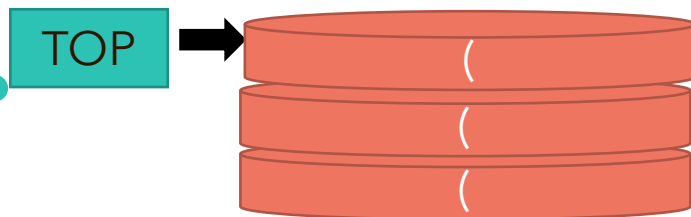
USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



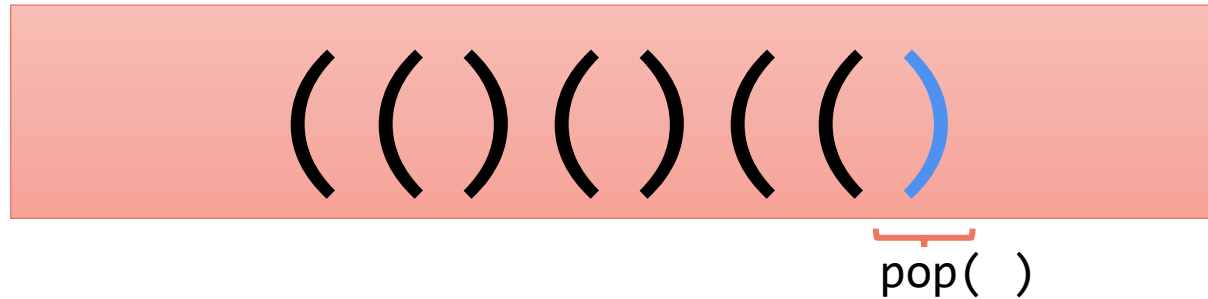
Si el carácter corresponde a un símbolo de apertura, entonces lo insertamos (PUSH) en el STACK



USO DE PILAS Y COLAS

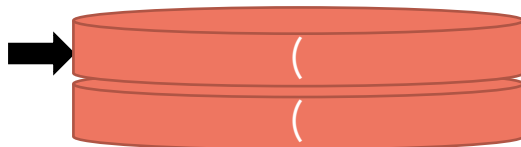
Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



Si el carácter corresponde a un símbolo de cierre, entonces eliminamos (POP) un carácter del STACK

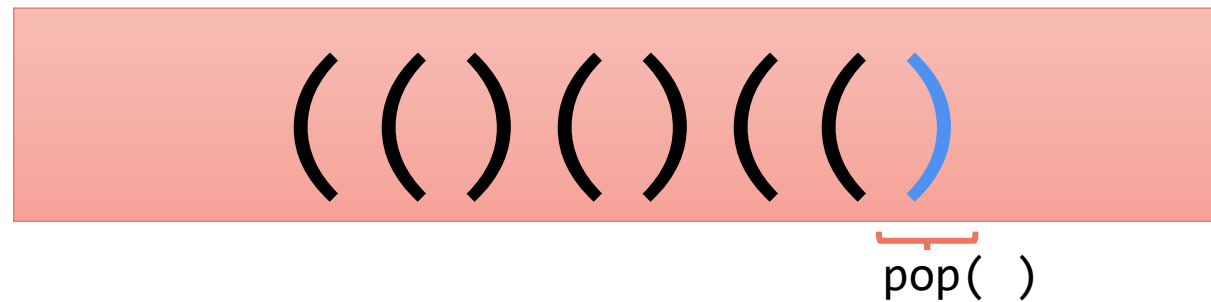
TOP



USO DE PILAS Y COLAS

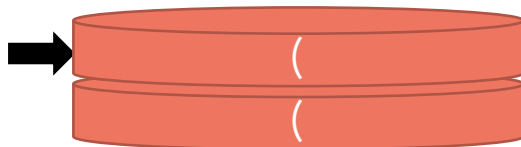
Problema 2: Balanceo de paréntesis

escribir un algoritmo que lea una cadena de paréntesis de izquierda a derecha y que decida si los símbolos están balanceados.



Al final, el STACK debe quedar vacío, sino la expresión esta desbalanceada

TOP



Problema 2: Balanceo de paréntesis

```
Balanceo(String P)
    Stack S = new Stack()
    for (int i=0; i<P.length; i++) //recorremos la cadena
        if P.charAt(i)=='(' //si abre, realizamos operación push
            S.push('(')
        else // realizamos operación pop, si el stack no esta vacio
            if S.isEmpty()
                return false
            else
                S.pop()
    if S.isEmpty() //verificamos que el stack quedo vacio
        return true
    else
        return false
```

USO DE PILAS Y COLAS

Problema 2: Balanceo de paréntesis



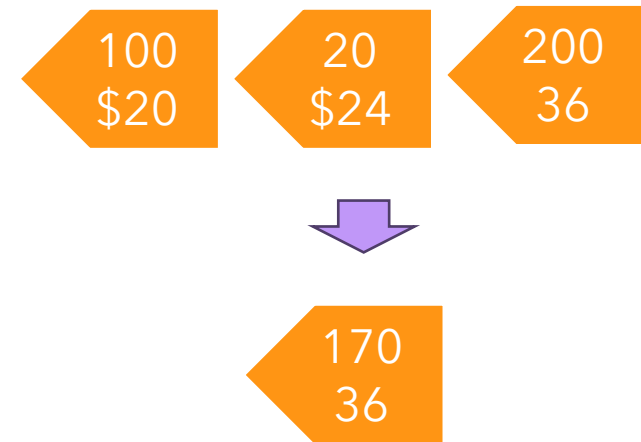
```
1 package hello;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6         System.out.println("Hello World!");
7     }
8
9 }
```

EXAMEN PARCIAL 2

Portafolio de inversión: cuando se vende una acción que hace parte de un portafolio de inversión, se calcula la ganancia o la pérdida, la cual resulta de la diferencia entre el precio de venta y el precio inicialmente pagado por la acción cuando se compró. Esta regla es sencilla de aplicar cuando solo se venden acciones adquiridas al mismo tiempo; sin embargo, en los portafolios de inversión es común encontrar acciones de la misma compañía compradas en diferentes periodos de tiempo; para lo cual es necesario mantener para cada compra el precio de la acción. Una forma sencilla de calcular la ganancia es emplear un protocolo FIFO (first-to-in first-to-out) en el cual se venden las acciones que llevan mas tiempo en el portafolio. Por ejemplo, suponga que se compraron 100 acciones a \$20 cada una el día 1, 20 acciones a \$24 en el día 2, 200 acciones a \$36 en el día 3, y se venden 150 acciones en el día 4 a \$30 cada una. El protocolo FIFO indica que para vender las 150 acciones se venderá 100 del día 1, 20 del día 2, y 30 del día 3; por tanto la ganancia por la venta de las acciones será $100 \cdot 10 + 20 \cdot 6 + 30 \cdot (-6) = \940 . Describa un algoritmo que dado una cola Q, donde se almacenan el número de acciones compradas y el valor de cada uno por día, el total de acciones a vender y el precio de venta de cada acción, retorne las ganancias producto de la venta (gananciaVenta(Queue Q, int numAcciones, int precioVenta):int).

EXAMEN PARCIAL 2

Por ejemplo, suponga que se compraron 100 acciones a \$20 cada una el día 1, 20 acciones a \$24 en el día 2, 200 acciones a \$36 en el día 3, y se venden 150 acciones en el día 4 a \$30 cada una. El protocolo FIFO indica que para vender las 150 acciones se venderá 100 del día 1, 20 del día 2, y 30 del día 3; por tanto la ganancia por la venta de las acciones será $100 \cdot 10 + 20 \cdot 6 + 30 \cdot (-6) = \940 . Describa un algoritmo que dado una cola Q, donde se almacenan el número de acciones compradas y el valor de cada uno por día, el total de acciones a vender y el precio de venta de cada acción, retorne las ganancias producto de la venta (`gananciaVenta(Queue Q, int numAcciones, int precioVenta):int`).



EXAMEN PARCIAL 2

```
gananciaVenta(Queue Q, int numAcciones, int precioVenta)
    if !Q.isEmpty()
        ganancia = 0
        numTemp = numAcciones
        while(!Q.isEmpty() & Q.first().getNoAcciones()<numTemp)
            temp = Q.dequeue()
            ganancia += temp.getNoAcciones()*(previoVenta-temp.getPrecio())
            numTemp = numTemp - temp.getNoAcciones()
    if Q.isEmpty()
        print("No se tenían suficientes acciones")
    elseif numTemp>0
        temp = Q.dequeue()
        ganancia += numTemp*(previoVenta-temp.getPrecio())
        temp.setNoAcciones(temp.getNoAcciones()-numTemp())
        Q2 = new Queue()
        Q2.enqueue(temp)
        while(!Q.isEmpty())
            Q2.enqueue(Q.dequeue())
        Q = Q2;
    return ganancia
```

Tiempo de computo: $O(n)$