Taller de Usos de las Pilas

Las pilas tienen muchos usos interesantes en la informática. Vamos a ver tres muy importantes.

1. Balanceo de delimitadores en expresiones matemáticos

Aunque los programadores sólo usan paréntesis para escribir expresiones aritméticas en Kotlin, los matemáticos utilizan paréntesis, corchetes (o *paréntesis cuadrados* []) y llaves ({}) para el mismo propósito. Estos delimitadores deben estar emparejados correctamente. Por ejemplo, un paréntesis abierto debe corresponder con su paréntesis cerrado correspondiente. Adicionalmente, un par de delimitadores no se deben intersectar. De esta manera, una expresión puede contener una secuencia de delimitadores como se muestra a continuación:

```
{[()()]()}
```

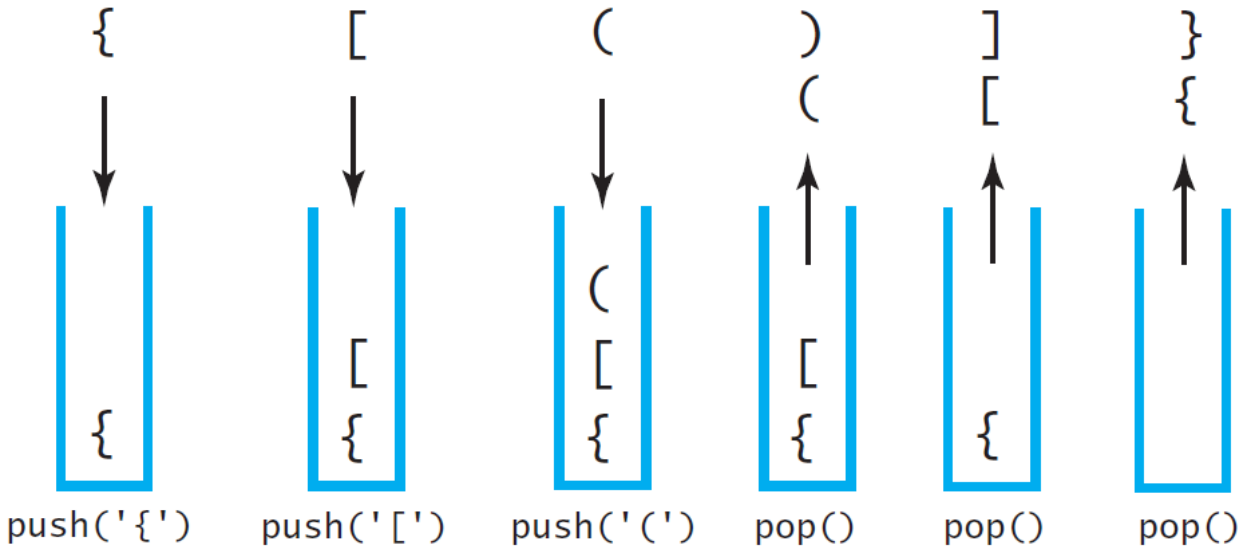
Pero la siguiente secuencia no está bien:

```
[(])
```

Por conveniencia, diremos que una **expresión balanceada** contiene delimitadores que se emparejan correctamente. Se requiere, entonces, crear un algoritmo que detecte si una expresión matemática está balanceada.

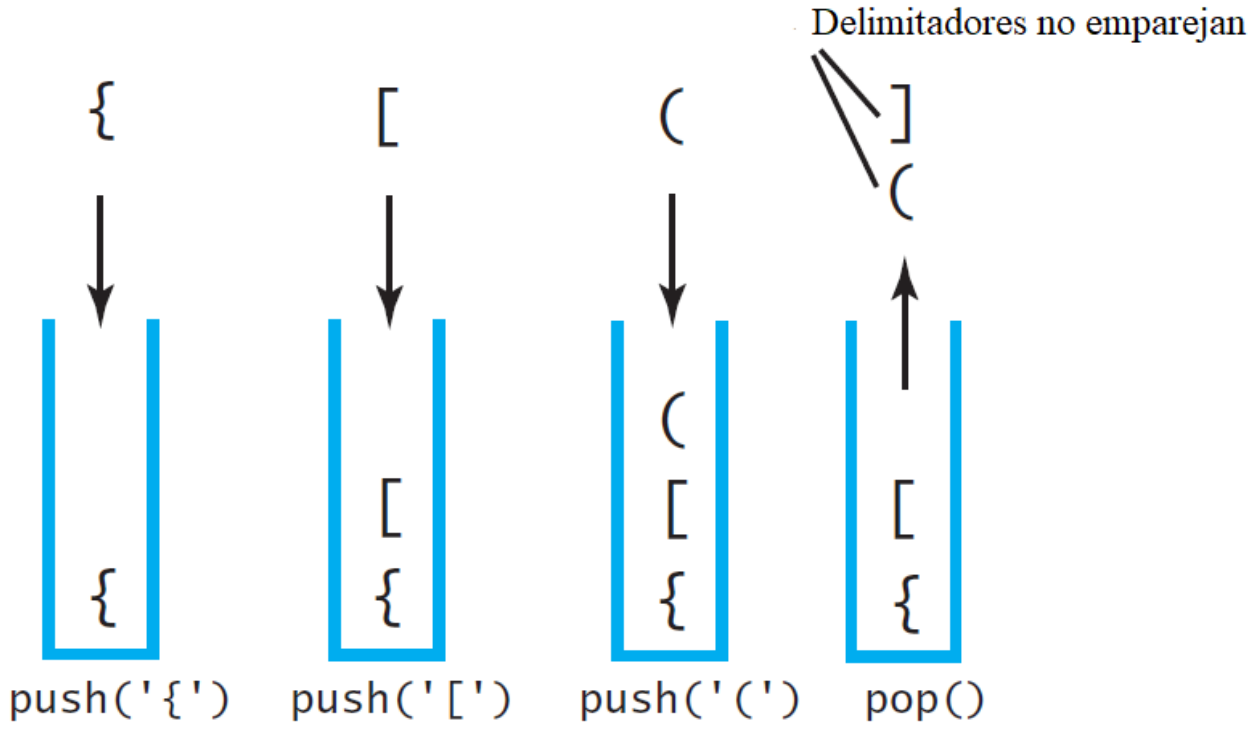
Por ejemplo, vamos a verificar si la siguiente expresión está balanceada:

$$a\ \{b\ [c\ (d\ +\ e)/2\ -\ f\ ]\ +\ 1\}$$

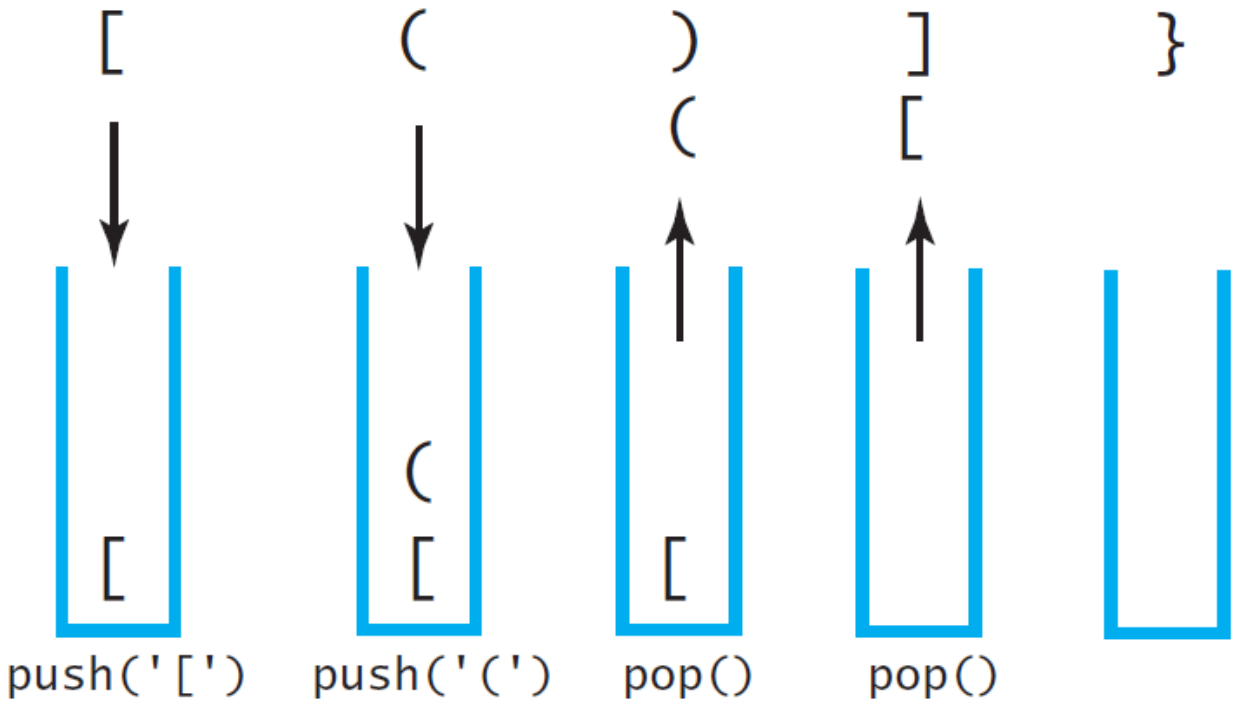


Como vemos, se usó una pila para ir almacenando los delimitadores a medida que los vamos encontrando. Al final, si no hay errores, podemos indicar que si está balanceada la expresión.

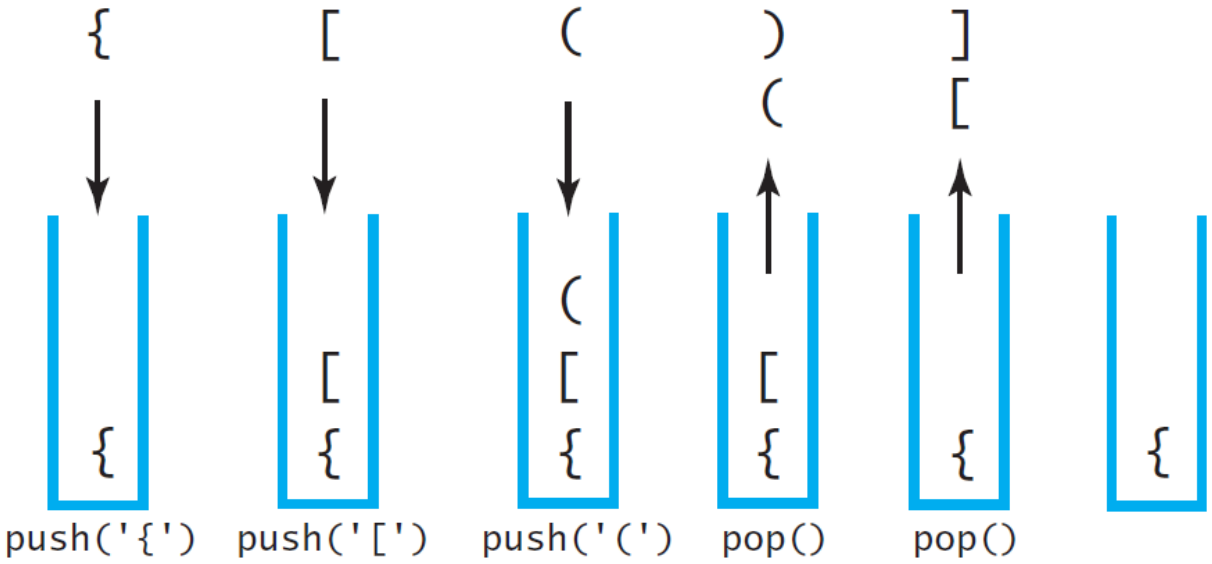
En cambio, la siguiente expresión no está balanceada, y el comportamiento del algoritmo se muestra a continuación: $\{\ a + [2 * (\ 3 + b\ ] - 1)\}$



Otros posibles errores de balanceo ocurren con la expresión [( )] }:



Como vemos, la última llave encuentra la pila vacía. Esto también indica que la expresión es incorrecta. Pero también es una situación anómala cuando encontramos una expresión como la siguiente: { [( )]



En este último caso, después de recorrer la expresión encontramos que la pila aún contiene información. Motivo por el cual la expresión no está balanceada.

El algoritmo sería como la siguiente:

```
Algoritmo está-balanceada(expresión)
    Crear una pila para llevar los delimitadores
    Para cada elemento en la expresión
        Si el elemento es un delimitador de apertura, o sea, "(", "[", "{" ent
            Coloque el elemento en el tope de la pila
        Sino, si el elemento es un delimitador de cierre ent
            Si la pila está vacía ent
                Retorne falso
            Fin si
            Obtenga y elimine el delimitador que está en el tope de la pila
            Si el elemento y el delimitador de la pila no son pareja ent
                Retorne falso
            fin si
        fin si
    fin para

    Si la pila no está vacía ent
        Retorne falso
    Sino
        Retorne cierto
    Fin si
Fin Algoritmo
```

2. Conversión de Infijo a Postfijo

El siguiente algoritmo permite pasar una expresión infija a postfija:

- Recorrer la expresión de principio a fin, obteniendo los diversos términos que hay en la lista. Por cada término hacer:
  - Si el término es un operador, introducirlo en la pila
  - Si el término es el paréntesis que abre "(", ignorarlo.
  - Si el término es el paréntesis que cierra ")", sacar el elemento del tope y agregarlo a la lista de salida.
  - Cualquier otro término se debe agregar a la lista de salida.

Vamos a trabajar con los siguientes operadores aritméticos: +, -, *, / y %.

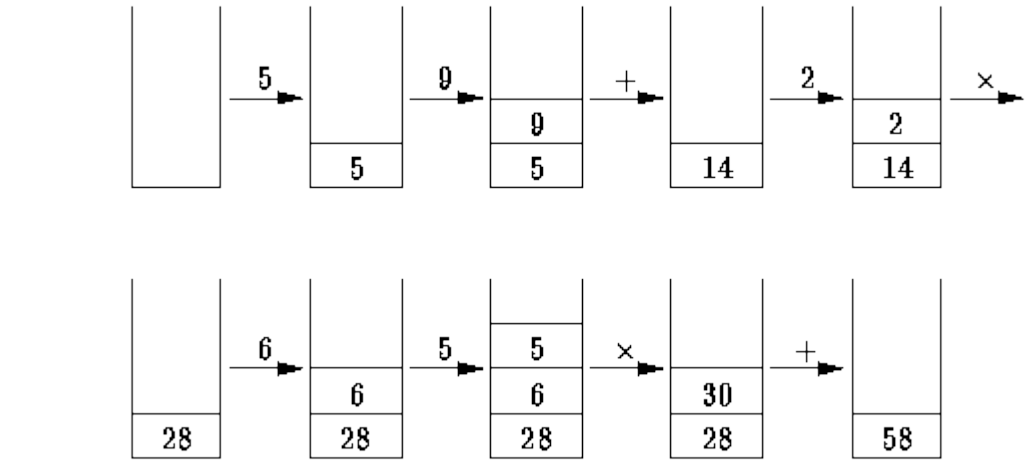
3. Evaluación de expresiones postfijas

Una interesante propiedad de la notación postfija (y prefija) es que no necesita de paréntesis para expresar el orden de aplicación de los operadores. Una expresión postfija se puede evaluar con la ayuda de una pila usando el siguiente procedimiento:

- Recorrer la expresión de izquierda a derecha y para cada termino hacer:
  - Sí el término es un número, introducirlo en una pila
  - Sí el término es un operador:
    - Sacar dos operandos de la pila
    - Aplicar el operador
    - Meter el resultado en la pila
- Al final, retorne el elemento que queda en el tope de la pila.

La siguiente figura ilustra el proceso:

$$5\,9\,+\,2\,\times\,6\,5\,\times\,+$$


5
9
+
2
×
9
5
14
14
6
5
×
+
6
5
6
30
28
28
28
28
58