Pilas

Especificación genérica

Pilas

Pilas

- Una pila es un tipo especial de lista en la que todas las inserciones y supresiones tienen lugar en un extremo denominado *Tope*.
- A las pilas se les llama también “Listas LIFO” (Last in first out) o listas “último en entrar, primero en salir”.
- Una pila es una versión restringida de una lista enlazada.
- A una pila se le pueden añadir y retirar nuevos nodos únicamente de su parte superior.

- El modelo intuitivo de una pila es precisamente una pila de fichas de póquer puesta sobre una mesa, o de libros sobre el piso, o de platos en una estantería, situaciones todas en las que sólo es conveniente quitar o agregar un objeto del extremo superior de la pila, al que se denominará en lo sucesivo “tope”.

- Un tipo de datos abstracto de la familia PILA incluye a menudo las cinco operaciones siguientes:

1. **ANULA(P)** convierte la pila en una pila vacía.
2. **TOPE(P)** devuelve el valor del elemento de la parte superior de la pila P.
3. **SACA(P)**, en inglés POP, suprime el elemento superior de la pila.

4. **METE(x, P)**, en inglés PUSH, inserta el elemento *x* en la parte superior de la pila P. el anterior tope se convierte en el siguiente elemento, y así sucesivamente.
5. **VACIA(P)** devuelve verdadero si la pila P esta vacía, y falso en caso contrario.

Implementación estática

- Para implementar una pila de manera estática mediante un arreglo se tiene en cuenta el hecho de que las inserciones y las supresiones ocurren solo en la parte superior.
- Se puede anclar la base de la pila a la base del arreglo y dejar que la pila crezca hacia la parte superior del arreglo.
- Una variable llamada tope indicará la posición actual del primer elemento de la pila

```
#define MAX 3

int main( )
{
    int tope = -1;
    int pila[MAX];

    push(pila, &tope, 1);
    push(pila, &tope, 2);
    push(pila, &tope, 3);

    while(!empty(tope))
    {
         valor = pop(pila, &tope);
         printf("%i\n",  valor);
    }
}
```

```
int full(int tope)
{
    if(tope == MAX-1)
         return 1;
    else
         return 0;
}
```

```
int empty(int tope)
{
    if (tope == -1)
         return 1;
    else
         return 0;
}
```

```
void push(int pila[], int* tope, int v)
{
    if(!full(*tope))
         pila[++(*tope)] = v;
    else
         printf("\nNo es posible agregar un elemento\n");
}
```

```
int pop(int pila[], int* tope)
{
    if(!empty(*tope))
         return (pila[(*tope)--]);
    else
         printf("\nNo es posible extraer un elemento\n");
}
```

Estructura Pila (LIFO)

- La pila (stack) es una estructura lineal sobre la que rigen ciertas restricciones a la hora de agregar o quitar elementos.

- A diferencia de las listas enlazadas en las que desarrollamos operaciones para insertar, agregar y eliminar nodos sin ningún tipo de limitación, decimos que la pila es una estructura lineal restrictiva de tipo LIFO(Last In First Out).

- Esto indica que el último elemento que ingresó a la pila debe ser el primero en salir.

Implementación de la estructura pila

- Implementaremos la pila sobre una lista enlazada para la cual solo desarrollaremos dos operaciones: poner (apilar un elemento) y sacar (obtener y eliminar un elemento de la pila).

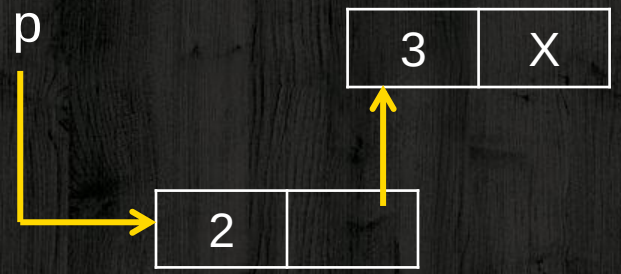
Operaciones poner (push) y sacar (pop)

- Supongamos que queremos poner en una pila los elementos del siguiente conjunto: {3, 2, 1}.

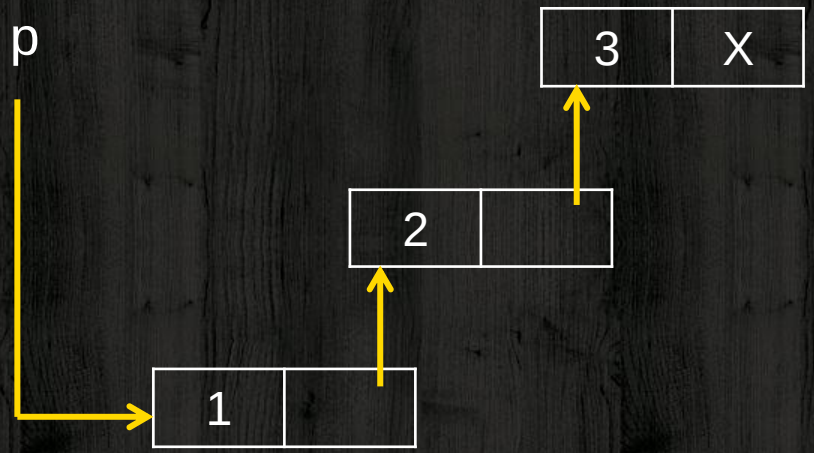
- Para esto, definimos un puntero **p** de tipo **Nodo*** inicializado en NULL y luego agregamos cada uno de los elementos del conjunto al inicio de la lista apuntada por **p.**

- Agregamos el 3 (primer elemento del conjunto):

- Agregamos el 2 (segundo elemento del conjunto):



- Agregamos el 1 (último elemento del conjunto):



- Luego, para sacar un elemento de la pila siempre tomaremos el primero de la lista.

Función Poner (Push)

```
void poner(Nodo** p, int v)
{
	Nodo* nuevo = (Nodo*) malloc(sizeof(Nodo));
	nuevo->valor = v;
	nuevo->sig = *p;
	*p = nuevo;
}
```

Función Sacar (Pop)

```
int sacar(Nodo** p)
{
	Nodo* aux = *p;
	int ret = aux->valor;

	*p = aux->sig;
	free(aux);

	return ret;
}
```

Notemos que al finalizar el programa no es necesario liberar la memoria ocupada por la pila porque la función ***sacar***, además de retornar el valor del elemento ubicado en la cima, desenlaza el nodo y lo libera con la función ***free*** de C.

Determinar si la pila tiene elementos o no

- La operación **pilaVacia** retorna ***true*** o ***false*** según la pila tenga o no elementos apilados.
- Esta operación es innecesaria ya que la pila estará vacía cuando el puntero al primer nodo de la lista sea NULL

```
int pilaVacia(Nodo* p)
{
     return p==NULL;
}
```

Recordemos que en C los valores booleanos se manejan con valores enteros.

Así, el número **0** (cero) equivale al valor booleano ***false*** y el número **1** (uno) o cualquier otro valor diferente de 0 equivale a ***true***.

Ejemplo de uso de una pila

- Se ingresa por teclado un conjunto de valores que finaliza con la llegada de un 0 (cero).
- Se pide mostrar los elementos del conjunto en orden inverso al original.
- Para resolver este programa utilizaremos una pila en la que apilaremos los valores a medida que el usuario los vaya ingresando.
- Luego, mientras que la pila no este vacía, sacaremos uno a uno sus elementos para mostrarlos por pantalla.

Aplicaciones de las pilas

1

Aplicaciones de las pilas

- Navegador Web – Se almacenan los sitios previamente visitados – Cuando el usuario quiere regresar (presiona el botón de retroceso), simplemente se extrae la última dirección (pop) de la pila de sitios visitados.

2

- Editores de texto – Los cambios efectuados se almacenan en una pila – Usualmente implementada como arreglo – Usuario puede deshacer los cambios mediante la operación “undo” o “deshacer”, la cual extrae el estado del texto antes del último cambio realizado.

3

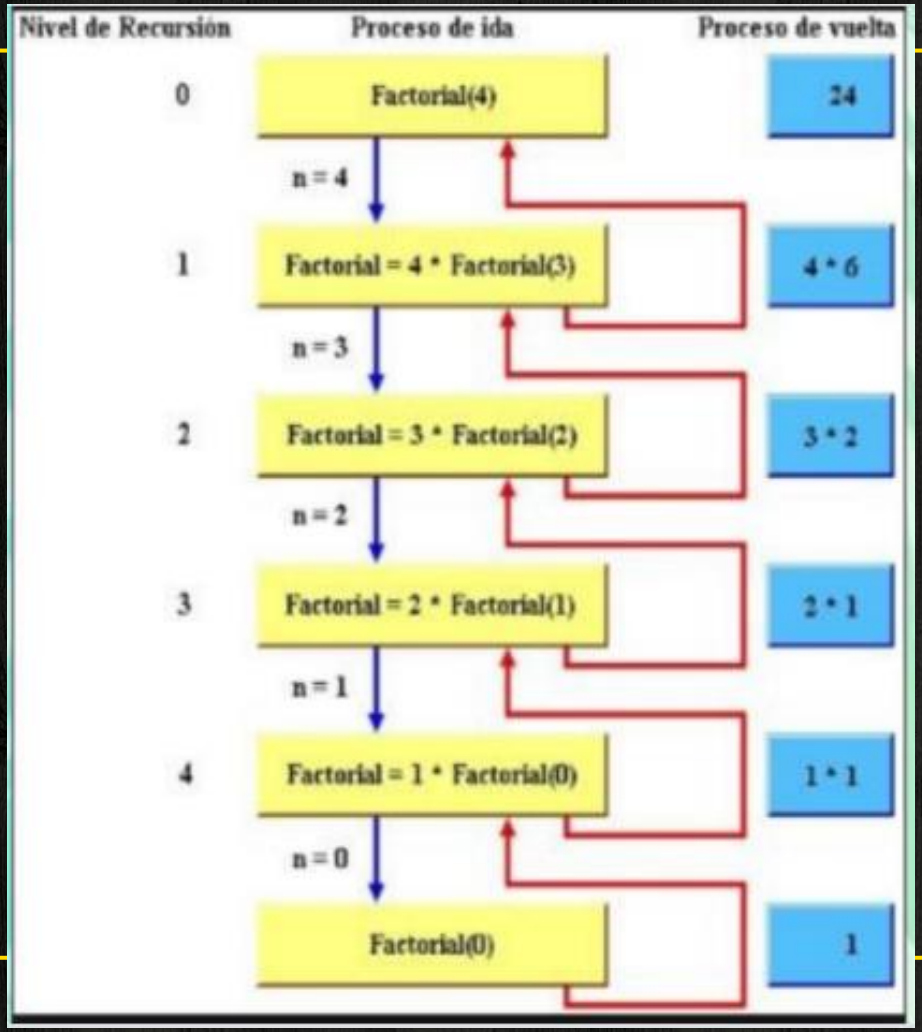
Llamadas a subprogramas

- Cuando se tiene un programa que llama a un subprograma, también conocido como módulo o función, internamente se usan pilas para guardar el estado de las variables del programa, así como las instrucciones pendientes de ejecución en el momento que hace la llamada.

- Cuando termina la ejecución del subprograma, los valores almacenados en la pila se recuperan para continuar con la ejecución del programa en el punto en el cual fue interrumpido. Además de las variables se recupera la dirección del programa la que se hizo la llamada, porque a esa posición se regresa el control del proceso.

4 Paso de programas recursivos a iterativos

- Para simular un programa recursivo es necesario la utilización de pilas, ya que se está llamando continuamente a subprogramas que a la vez vuelven a llamarse a si mismo.
- Veamos cómo se utiliza esta eliminación de la recursión mediante un ejemplo sencillo hecho en clase. El cálculo del número factorial.


Nivel de Recursión
Proceso de ida
Proceso de vuelta
0
Factorial(4)
24
n = 4
1
Factorial = 4 * Factorial(3)
4 * 6
n = 3
2
Factorial = 3 * Factorial(2)
3 * 2
n = 2
3
Factorial = 2 * Factorial(1)
2 * 1
n = 1
4
Factorial = 1 * Factorial(0)
1 * 1
n = 0
Factorial(0)
1

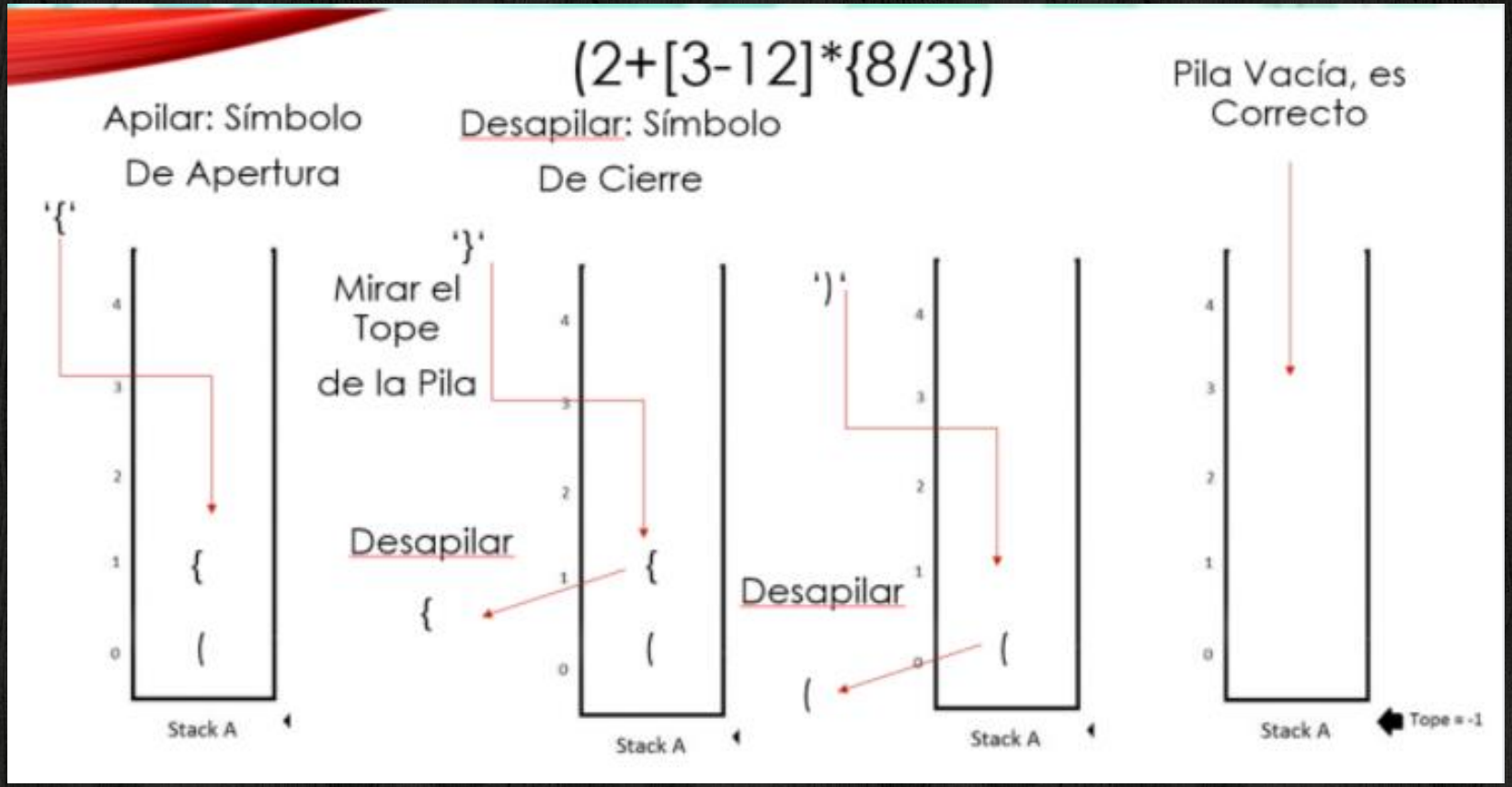
5

Tratamiento de expresiones aritméticas

- Es útil poder detectar si es que los paréntesis en un archivo fuente están o no balanceados.
- Se puede usar un stack: **a+(b+c)*([d+e])/f**

- Pseudocódigo - Crear el stack.
- Mientras no se ha llegado al final del archivo de entrada:
  - Descartar símbolos que no necesiten ser balanceados.
  - Si es un paréntesis de apertura: poner en el stack.
  - Si es un paréntesis de cierre, efectuar un pop y comparar.
  - Si son de igual tipo continuar
  - Si son de diferente tipo: avisar el error.
  - Si se llega al fin de archivo, y el stack no esta vacío: avisar error

(2+[3-12]*{8/3})


Apilar: Símbolo
De Apertura
'{'
4
3
2
1
0
{
(
Stack A
Desapilar: Símbolo
De Cierre
'}'
Mirar el
Tope
de la Pila
Desapilar
{
{
(
Stack A
')'
Desapilar
(
(
Stack A
Pila Vacía, es
Correcto
Stack A
Tope = -1

6

Tratamiento de Expresiones aritméticas

Convertir expresiones Infijas a Postfijas

- **1. A+B = Infija**
- **2. AB+ = posfija**
- **3. +AB = Prefija**

- **A+B = Infija:** Esta notación es Infija porque el operador se encuentra entre los operadores.
- **AB+ = posfija:** Esta notación es Posfija porque el operador se encuentra después de los operadores.
- **+AB = Prefija:** Esta es una notación Prefija porque el operador se encuentra antes de los operadores.

- EXPRESIONES ARITMÉTICAS: Una expresión aritmética contiene constantes, variables y operaciones con distintos niveles de precedencia.

- OPERACIONES Y PRECEDENCIA:

^	Potencia
* /	Multiplicación, División
+ -	Suma, Resta

- Para convertir una expresión dada en notación infija a una en notación postfija se establecen primero ciertas condiciones:
  - Los operadores de más alta prioridad se ejecutan primero.
  - Si hubiera en una expresión dos o más operadores de igual prioridad, entonces procesarán de izquierda a derecha.
  - Las subexpresiones que se encuentran entre paréntesis tendrán más prioridad que cualquier operador.

Expresión infija: X + Z * W
Expresión posfija: X Z W * +

1. El primer operador que se procesa durante la traducción de la expresión es la multiplicación, paso 1, debido a que es el de más alta prioridad.
2. Se coloca el operador tal manera que los operandos afectados por él lo precedan.
3. Para el operador de suma se sigue el mismo criterio, los dos operandos lo preceden. En este caso el primer operando es X y el segundo es ZW*.

Paso	Expresión
0	X + Z * W
1	X + Z W *
2	X Z W * +

Expresión infija: (X + Z)* W / T ^ Y – V
Expresión postfija: X Z + W * T Y ^ / V -

1. En el paso 1 se convierte la subexpresión que se encuentra entre paréntesis por ser la de más alta prioridad. Luego se sigue con el operador de potencia, paso 2,
2. Y así sucesivamente con los demás, según su jerarquía.
3. Como consecuencia de que la multiplicación y la división tienen igual prioridad, se procesa primero la multiplicación por encontrarse más a la izquierda en la expresión, paso 3.
4. El operador de la resta es el último que se mueve, paso 5.

Paso	Expresión
0	(X + Z) * W / T ^ Y - V
1	X Z + * W / T ^ Y - V
2	X Z + * W / T Y ^ - V
3	X Z + W * / T Y ^ - V
4	X Z + W * T Y ^ / - V
5	X Z + W * T Y ^ / V -

Algoritmo para convertir expresión infija a postfija

- Se crea un string resultado donde se almacena la expresión en postfijo.
  1. Los operandos se agregan directamente al resultado
  2. Un paréntesis izquierdo se mete a la pila y tiene prioridad o precedencia cero (0).
  3. Un paréntesis derecho saca los elementos de la pila y los agrega al resultado hasta sacar un paréntesis izquierdo.
  4. Los operadores se insertan en la pila si:
     a) La pila esta vacía.
     b) El operador en el tope de la pila tiene menor precedencia.
     c) Si el operador en el tope tiene mayor o igual precedencia se saca y agrega al resultado (repetir esta operación hasta encontrar un operador con menor precedencia o la pila este vacía).
  5. Cuando se termina de procesar la cadena que contiene la expresión infijo se vacía la pila pasando los elementos al resultado.

En los pasos 1, 3 Y5 el símbolo analizado -un operando- se agrega directamente a EPOS. Al analizar el operador +, paso 2, se verifica si en PILA hay operadores e mayor o igual prioridad.

En este caso, PILA está vacía; por tanto, se pone el símbolo en el tope de ella. Con el operador *, paso 4, sucede algo similar.

En PILA no existen operadores de mayor o igual prioridad -la suma tiene menor prioridad que la multiplicación-, por lo que se agrega el operador * a PILA.
En los dos últimos pasos, 6 y 7, se extraen de PILA sus elementos, agregándolos a EPOS.

Expresión infija: X + Z * W

Paso	*EI*	Símbolo analizado	Pila	EPOS
0	*X + Z * W*			
1	*+ Z * W*	*X*		*X*
2	*Z * W*	+	+	*X*
3	** W*	*Z*	+	*XZ*
4	*W*	*	+ *	*XZ*
5		*W*	+ *	*XZW*
6			+	*XZW* *
7				*XZW* * +

Los pasos que se consideran más relevantes son: en el paso 5, al analizar el paréntesis derecho se extraen repetidamente todos los elementos de PILA (en este caso sólo el operador +), agregándolos a EPOS hasta encontrar un paréntesis izquierdo.

El paréntesis izquierdo se quita de PILA pero no se incluye en EPOS -recuerde que las expresiones en notación posfija no necesitan de paréntesis para indicar prioridades.

Cuando se trata el operador de división, paso 8, se quita de PILA el operador * y se agrega a EPOS, ya que la multiplicación tiene igual prioridad que la división.

Al analizar el operador de resta, paso 12, se extraen de PILA y se incorporan a EPOS todos los operadores de mayor o igual prioridad, en este caso son todos los que están en ella –la potencia y la división-, agregando finalmente el símbolo en PILA.

Luego de agregar a EPOS el último operando, y habiendo revisado toda la expresión inicial, se vacía PILA y se incorporan los operadores (en este caso el operador -) a la expresión postfija.

Paso	*EI*	Símbolo analizado	Pila	EPOS
0	$(X+Z)*W/T\wedge Y-V$			
1	$X+Z)*W/T\wedge Y-V$	(	(	
2	$+Z)*W/T\wedge Y-V$	X	(	X
3	$Z)*W/T\wedge Y-V$	+	(+	X
4	$)*W/T\wedge Y-V$	Z	(+	XZ
5	$*W/T\wedge Y-V$	)	(	$XZ+$
		)		$XZ+$
6	$W/T\wedge Y-V$	*	*	$XZ+$
7	$/T\wedge Y-V$	W	*	$XZ+W$
8	$T\wedge Y-V$	/	/	$XZ+W*$
		/	/	$XZ+W*$
9	$\wedge Y-V$	T	/	$XZ+W*T$
10	$Y-V$	$\wedge$	/^	$XZ+W*T$
11	$-V$	Y	/^	$XZ+W*TY$
		–	/	$XZ+W*TY\wedge$
12	V	–		$XZ+W*TY\wedge/$
		–	–	$XZ+W*TY\wedge/$
13		V	–	$XZ+W*TY\wedge/V$
14				$XZ+W*TY\wedge/V-$

7

Notación Postfija

Notación Polaca Inversa
Reverse Polish Notation (RPN)

- Es un método algebraico alternativo de introducción de datos.
- Su principio es el de evaluar los datos directamente cuando se introducen y manejarlos dentro de una estructura de tipo Pila **LIFO** (Last In First Out), lo que optimiza los procesos a la hora de programar.
- En la **notación postfija** el operador va después de los operandos.

Notación postfija

5 + 2	Infija
+ 5 2	Prefija
5 2 +	Postfija

Infija	Postfija	Resultado
2 + 3 * 4	2 3 4 * +	14
(2 + 3) * 4	2 3 + 4 *	20
5+((1+2)*4)-3	5 1 2 + 4 * + 3 -	14

Algoritmo de Notación Polaca Inversa (RPN)

- Si hay elementos en la bandeja de entrada:
  - Leer el primer elemento de la bandeja de entrada.
  - Si el elemento es un operando.
    - Poner el operando en la pila.
  - Si no, el elemento es un operador.
    - Se sabe que el operador toma 2 operandos.
    - Si hay menos de 2 operandos en la pila.
      - **(Error)** El usuario no ha introducido suficientes argumentos en la expresión.
    - Si no, tomar los últimos 2 operandos de la pila.
    - Evaluar la operación con respecto a los operandos.
    - Introducir el resultado en la pila.
- Si hay un solo elemento en la pila:
  - El valor de ese elemento es el resultado del cálculo.
- Si hay más de un elemento en la pila:
  - **(Error)** El usuario ha introducido demasiados elementos.

Algoritmo de Notación Polaca Inversa (RPN)

La expresión algebraica 5+((1+2)*4)-3 se traduce a la notación polaca inversa como **5 1 2 + 4 * + 3 -** y se evalúa de izquierda a derecha según se muestra en la siguiente tabla.

Entrada	Operación	Pila	Comentario
5	Introducir en la pila	5	
1	Introducir en la pila	5, 1	
2	Introducir en la pila	5, 1, 2	
+	Suma	5, 3	Tomar los dos últimos valores de la pila (1, 2), hacer 1 + 2 y sustituirlos por el resultado (3)
4	Introducir en la pila	5, 3, 4	
*	Multiplicación	5, 12	Tomar los dos últimos valores de la pila (3, 4), hacer 3 * 4 y sustituirlos por el resultado (12)
+	Suma	17	Tomar los dos últimos valores de la pila (5, 12), hacer 5 + 12 y sustituirlos por el resultado (17)
3	Introducir en la pila	17, 3	
−	Resta	14	Tomar los dos últimos valores de la pila (17, 3), hacer 17 - 3 y sustituirlos por el resultado (14)