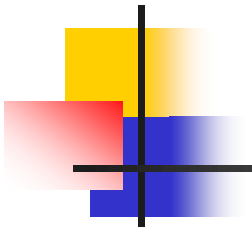


Índice

- TDA Pila
  - Definición y operaciones básicas
  - Operaciones e implementación
  - Aplicaciones

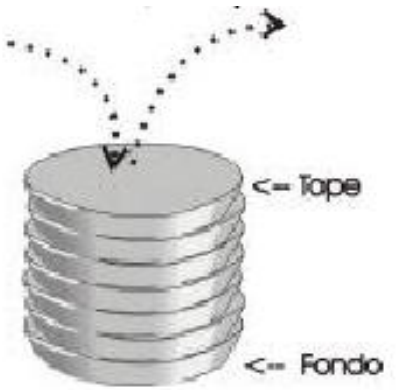
TDA PILA

Definición

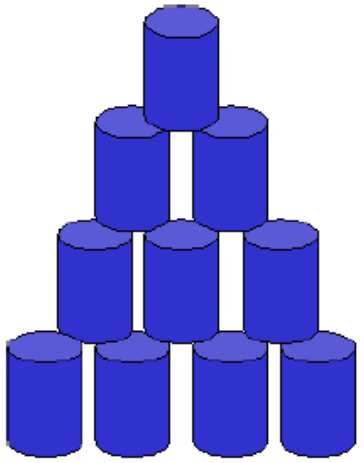
Def: una pila es una lista ordenada de elementos en la que todas las inserciones y supresiones se realizan por un mismo extremo denominado tope o cima de la pila.

Estructura LIFO (*Last In First Out*):

"último en entrar primero en salir"

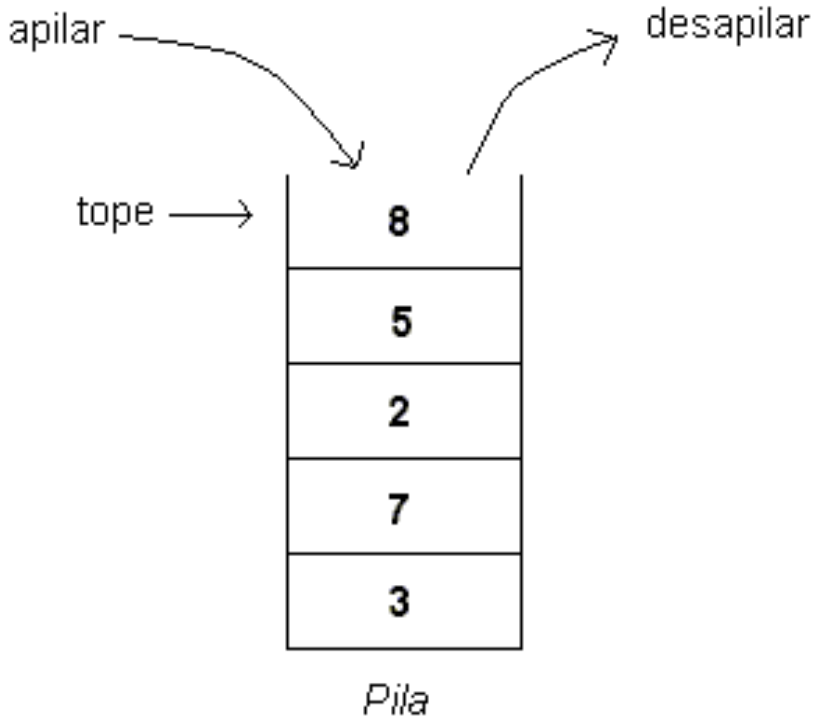


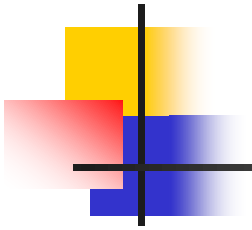
Pila de latas en un supermercado



Operaciones básicas

- PUSH: apilar, meter
- POP: desapilar, sacar
- TOP: cima, tope





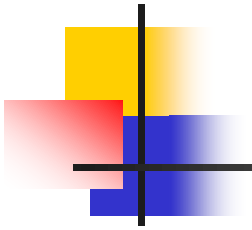
Operaciones

-**PUSH (insertar).-** Agrega un elementos a la pila en el extremo llamado **tope**.

-**POP (remover).-** Remueve el elemento de la pila que se encuentra en el extremo llamado **tope**.

-**VACIA.-** Indica si la pila contiene o no contiene elementos.

-**LLENA.-** Indica si es posible o no agregar nuevos elementos a la pila.



Implementación

Vectores

- Variables estáticas
- Tamaño máximo fijo
  - Peligro de desbordamiento (*overflow*)
  - Uso ineficiente de memoria

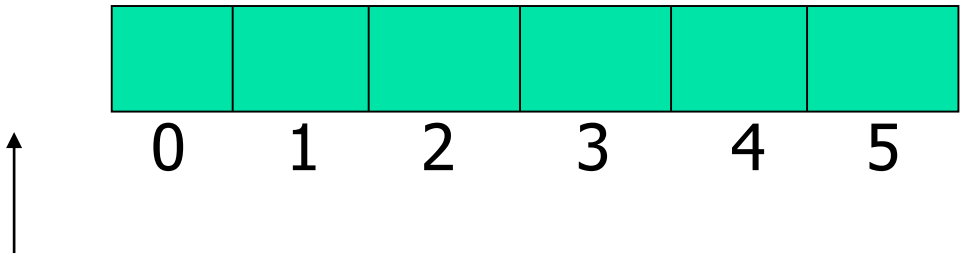
Listas enlazadas

- Variables dinámicas
- No riesgo de *overflow*
- Limitadas por memoria disponible
- Cada elemento necesita más memoria (guardar dirección siguiente)
- Uso eficiente de memoria

Problema común: *underflow* o subdesbordamiento

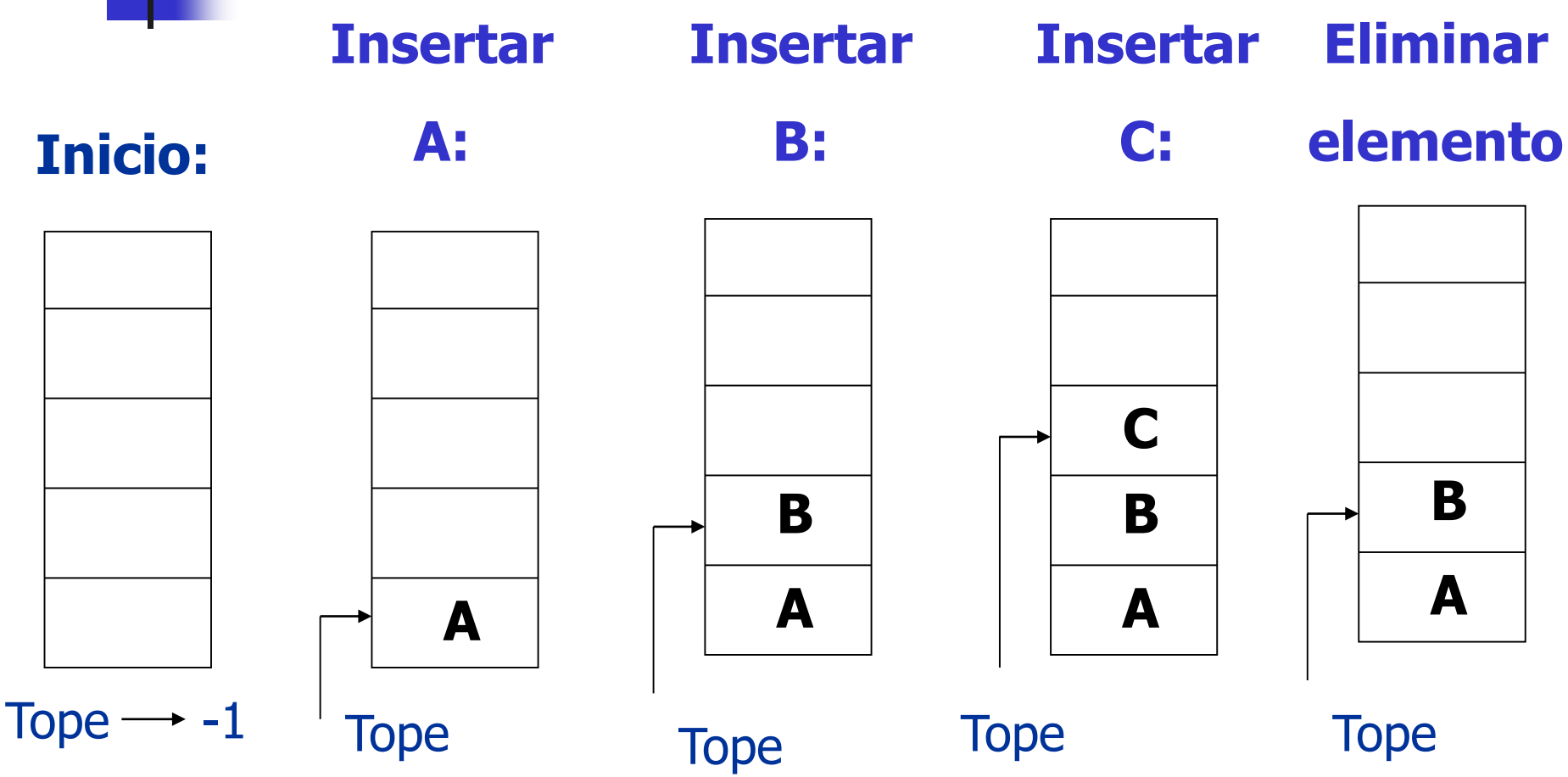
REPRESENTACIÓN DE PILAS:

- Usando arreglos: Define un arreglo de una dimensión (vector) donde se almacenan los elementos.



TOPE: Apunta hacia el elemento que se encuentra en el extremo de la pila. (inicialmente es -1). (PILA VACIA)

Ejemplo



Pila Llena cuando Tope=No. máximo de casillas disponibles

Implementación con vectores

- Definición de tipos

```
ELEMENTO = T;
PILA = registro de
    tope: numérico;
    arreglo: vector[1..MAX] de
    ELEMENTO;
finregistro;
```

- Operación Push

```
Algoritmo PUSH (P: Pila, X:
    ELEMENTO, ok: lógico) es
resp: lógico;
INICIO
    Llena?(P,resp);
    si resp entonces
            ok:= falso;
    sino
            P.tope:= P.tope + 1;
            P.arreglo[P.tope]:= X;
            ok:= cierto;
    finsi;
FIN
```

Implementación con vectores

- Operación Pop

```
Algoritmo POP (P: PILA, X:
    ELEMENTO, ok: lógico) es
INICIO
    Vacia(P, resp);
    si resp entonces
            ok:= falso;    {no hay
                              elementos
    q sacar}
    sino
            X:= P.arreglo[P.tope];
            P.tope:= P.tope -1;
            ok:= cierto;
    finsi;
FIN
```

- Operación Top

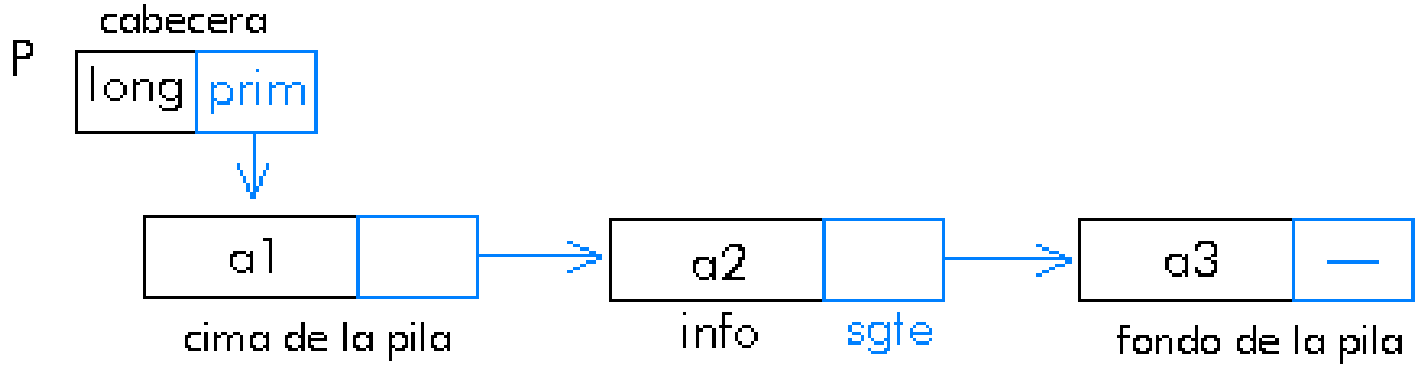
```
Algoritmo TOP (P: PILA, X:
    ELEMENTO, ok:lógico) es
resp: lógico;
INICIO
    Vacia?(P, resp);
    si resp entonces
            ok:= falso; {pila vacía}
    sino
            ok:= cierto;
            X:= P.arreglo[P.tope];
    finsi;
FIN
```

Implementación con listas enlazadas

- Definición de tipos

```
ELEMENTO = T;
NODO = registro de
    info: ELEMENTO;
    sgte: puntero a NODO;
finregistro;
POSICION = puntero a NODO;
```

```
PILA = registro de
       longitud: numerico;
       prim: POSICIÓN;
finregistro;
```



Implementación con listas enlazadas

- Operación Push

```
Algoritmo PUSH (P: PILA, X: ELEMENTO, ok: logico) es
resp: logico;
temp: POSICION;
INICIO
    Llena?(P,resp); {resp=falso si no se puede reservar más memoria}
    si resp entonces
            ok := falso;
            Escribir "Pila llena";
    sino
            Obtener(temp);
            temp→.info := X;
            temp→.sgte := P.prim; {será nil si la pila estaba vacía}
            P.prim := temp;
            P.longitud := P.longitud +1;
            ok := cierto;
    finsi
FIN
```

Implementación con listas enlazadas

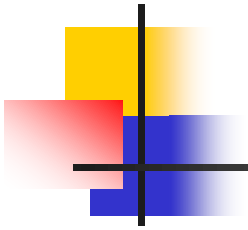
Operación Pop

```
Algoritmo POP (P: PILA, X: ELEMENTO, ok: logico) es
resp: lógico;
temp: POSICION;
INICIO
      Vacia?(P, resp);
      si resp entonces
       ok := falso; {la pila está vacía}
      sino   {procedemos a sacar el último elemento insertado}
       temp := P.prim;
       P.prim := temp→.sgte; {que será nil si sólo hay un elemento en la pila}
       X := temp→.info;
       Liberar(temp);
       ok := cierto;
      finsi;
FIN
```

Implementación con listas enlazadas

- Operación Top

```
Algoritmo TOP(P: PILA, X: ELEMENTO, ok: lógico) es
resp: lógico;
INICIO
    Vacia?( P, resp);
    si resp entonces
            ok:= falso; {Pila vacia}
    sino
            X := P.prim→.info;
            ok:= cierto;
    finsi;
FIN
```

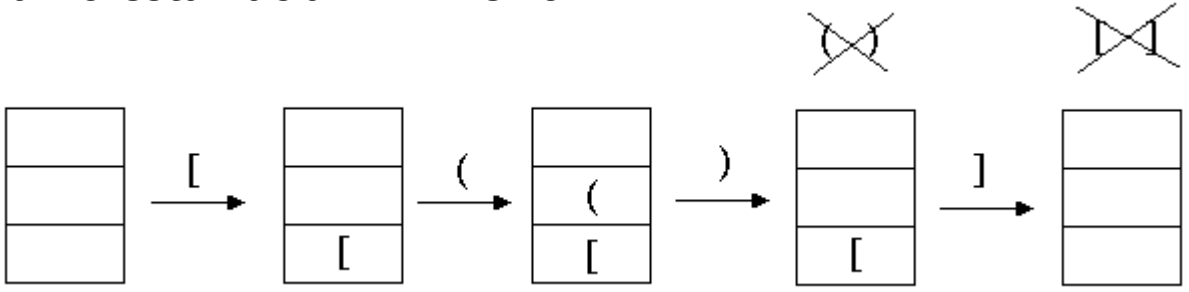


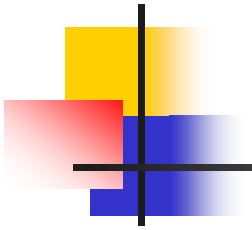
Aplicaciones de las pilas

- Gran uso en compiladores y SO's.
- Entornos donde haya que recuperar el último valor que se almacenó (*backtracking*)
- Algunas aplicaciones:
  - Equilibrado de símbolos
  - Llamadas a subprogramas
  - Eliminación de recursividad
  - Tratamiento de expresiones aritméticas
    - Evaluación de expresiones postfijas
    - Conversión infija a postfija
  - Borrado de caracteres en un editor de textos

Equilibrado de símbolos

- Se van leyendo los caracteres. Cuando se encuentra un elemento clave (paréntesis, corchete…) se trata según su tipo:
  - Si es de apertura: se mete en la pila.
  - Si es de cierre:
    - Si la pila está vacía ⟶ error.
    - Si la pila no está vacía:
      - Si la cima es el correspondiente símbolo de apertura se extrae.
      - Si no lo es ⟶ error.
- Si al final la pila no está vacía ⟶ error





Llamadas a subprogramas

- Al llamar a un subprograma se necesita guardar:
  - Estado de las variables locales del programa que llama
  - Dirección del programa en la que se hizo la llamada

  } *Registro de activación*
- Al hacer la llamada esta información se mete en la cima de una pila.
- Al terminar el subprograma, se saca la cima y se recupera el estado del momento de la llamada vuelve al punto de ejecución donde se hizo la llamada.
- El subprograma puede llamar a otros subprogramas y así sucesivamente.
- Permite implementar la recursión.

Peligro: rebasamiento de la pila

Aplicaciones de Pilas

Control de secuencia de programas.

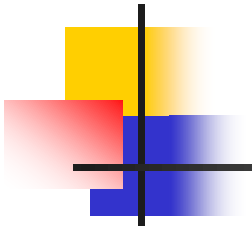
- Las pilas son requeridas para implementar el control de flujo de ejecución de un programa con subprogramas (funciones, procedimientos o métodos).
  - Subprogramas recursivos o no recursivos
  - Existen *llamadas a ejecución* de subprogramas.
  - Un subprograma *se ejecuta completamente* antes de retornar al punto

Aplicaciones de Pilas

Control de secuencia de programas.

```
// Programa Principal
class Principal{
   public static void proceso1(){
      System.out.println("proceso1");
      proceso2();
      …
   }
   public static void proceso2(){
     System.out.println("proceso2");
      …
   }
   public static void main(String[] args){
       proceso1();
       proceso2();
       …
   }
}
```

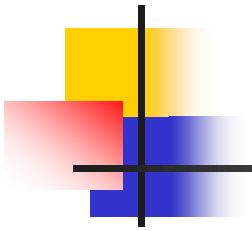
Cual es la salida de este programa?



Eliminación de recursividad

- La recursión consume muchos recursos (memoria).
- Recursión de cola: la llamada recursiva está en la última línea. Para eliminarla:
  - Se necesita: argumentos del algoritmo pasados por valor o referencia si son los mismos argumentos los que se pasan a la llamada recursiva.
  - Se asignan a los argumentos los valores que se van a pasar en la llamada recursiva.
  - Salto (*goto*) al principio de la rutina.
- Para transformar algoritmo recursivo en iterativo:
  - Se guarda en pilas el estado del problema en el momento de la llamada recursiva.
  - Se vuelve al principio de la rutina mediante una estructura iterativa.
  - La vuelta atrás de la recursión se consigue sacando los valores de las pilas

Aplicaciones de Pilas



Funciones Recursivas

- Las pilas pueden ser usadas para implementar la recursión en programas.
- Una función o procedimiento recursivo es aquel que se llama a si mismo.
- Ejemplos:
  - Factorial
  - Números de Fibonacci
  - Torres de Hanoi
  - Algoritmos de Ordenamiento de datos
  - Etc.

Aplicaciones de Pilas

Recursion

```
// Funcion factorial
   public static int factorial(int n) {
       if (n<=1) return 1;
       else return n*factorial(n-1);
   }
```

```
// Funcion fibonacci
   public static int fib(int n) {
       if (n==1) return 0;
       else if (n==2) return 1;
              else return fib(n-1)+fib(n-2);
   }
```

EXPRESIONES ARITMETICAS:

Una expresión aritmética contiene constantes, variables y operaciones con distintos niveles de precedencia.

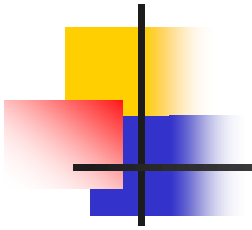
OPERACIONES :

^ potencia

*/ multiplicación, división

+,- suma, resta

NOTACIONES:



NOTACION INFIJA:

Los operadores aparecen en medio de los operandos.

A + B, A – 1, E/F, A * C , A ^ B , A + B + C, A+B-C

NOTACION PREFIJA:

El operador aparece antes de los operandos.

+ AB, - A1, /EF, *AC, ^AB, +AB+C, +AB-C

NOTACION POSTFIJA:

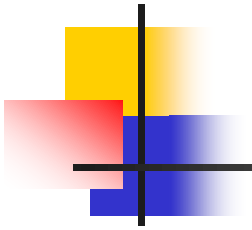
El operador aparece al final de los operandos.

AB+, A1-, EF/, AC*, AB^, AB+C+, AB+C-

PASOS PARA EVALUAR UNA EXPRESION:

1.-CONVERTIR A POSTFIJO: convertir la expresión en notación infijo a notación postfijo

2.-EVALUAR LA EXPRESION POSTFIJA: usar una pila para mantener los resultados intermedios cuando se evalúa la expresión en notación posfijo.



Tratamiento de expresiones aritméticas

- Notación infija: a + b
- Notación prefija: + a b
- Notación postfija: a b +

- **Problema**: distinción de prioridades en notación infija.
  Ej: evaluar a + b * c
- **Soluciones**:
  - Empleo de paréntesis.
  - Conversión a notación prefija o postfija.

- Ventajas de la notación postfija:
  - No hace falta conocer reglas de prioridad.
  - No hace falta emplear paréntesis.

REGLAS PARA CONVERTIR EXPRESION 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:

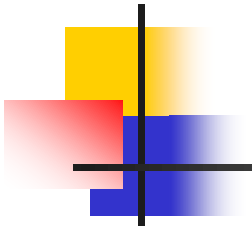
a) La pila esta vacía.

b) El operador en el tope de la pila tiene menor precedencia.

c) Si el operador en el tope tiene mayor 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**.

Aplicaciones de las pilas



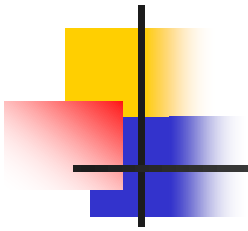
Tratamiento de expresiones aritméticas:
Conversión infija a postfija

- **Operandos:** se colocan directamente en la salida.

- **Operadores:** si la pila está vacía, lo metemos en la pila. Si no:
  - Si en la cima se encuentra un operador de menor prioridad: push()
  - Si no: pop() hasta que en la cima haya uno de menor prioridad, un paréntesis de apertura o la pila esté vacía. Entonces se hace un push().

- **Paréntesis:**
  - de apertura '(' : se mete en la pila.
  - de clausura ')' : se van llevando los operadores de la pila a la salida hasta que se encuentra uno de apertura, que se saca de la pila.

- Para finalizar, si en la pila aún queda algún operador, se lleva a la salida.



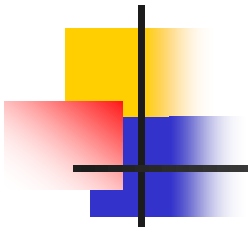
Ejemplos

- Convertir las siguientes expresiones infijas a posfijo

A +B*C-D

A * ((B-C) / 2))

((X-Z)*(Y+W))/X+Y



REGLAS PARA EVALUAR UNA EXPRESION POSTFIJA

Recorrer la expresión de izquierda a derecha

1. Si es un operando
   1. almacenar el valor en la pila de valores
2. Si es un operador:
   1. Obtener dos operandos de la pila de valores
   2. Aplicar el operador
   3. Almacenar el resultado en la pila de valores

Al finalizar el recorrido, el resultado estará en la pila de valores

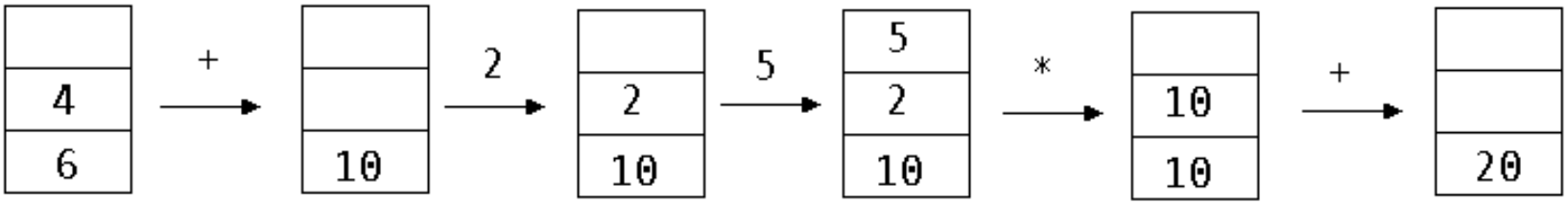
Aplicaciones de las pilas

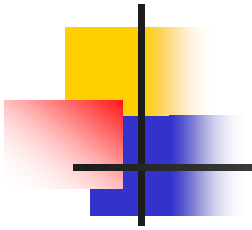
Tratamiento de expresiones aritméticas:

Evaluación de expresiones postfijas

- Se lee la expresión elemento a elemento.
- Si es un operando se mete en una pila
- Si es un operador, se extraen los dos últimos elementos introducidos en la pila, se aplica el operador sobre ellos y el resultado se guarda en la pila.

Ejemplo: 6 4 + 2 5 * +



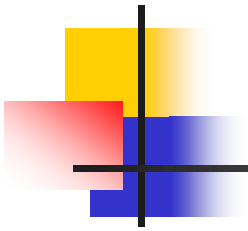


Borrado de caracteres en un editor de texto

- Se van leyendo los caracteres de uno en uno.
- Si el carácter no es de borrado ni de eliminación de línea, se mete en la pila.
- Si el carácter es de borrado, se hace un *pop()*, para sacar el elemento cima de la pila.
- Si el carácter es de eliminación de línea se vacía toda la pila.

Ejemplo: si # es el carácter de borrado,

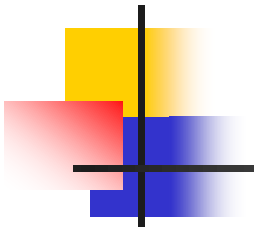
i n p # f o r o n # # m á t i z # c a ⟶ *informática*



Clase Stack en Java

La clase **Stack** representa una pila de objetos donde el último en entrar es el primero en salir (LIFO). Extiende la clase **Vector** con 5 operaciones básicas.

java.util
Class Stack<E>
java.lang.Object
    java.util.AbstractCollection<E>
        java.util.AbstractList<E>
            java.util.Vector<E>
                java.util.Stack<E>
All Implemented Interfaces:
    Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess



Method Summary	
boolean	empty() Tests if this stack is empty.
E	peek() Looks at the object at the top of this stack without removing it from the stack.
E	pop() Removes the object at the top of this stack and returns that object as the value of this function.
E	push(E item) Pushes an item onto the top of this stack.
int	search(Object o) Returns the 1-based position where an object is on this stack.