Técnico Informático

ESTRUCTURA DE DATOS

UNIDAD III: TDA PILA

Escuela de Minas "Dr. Horacio Carrillo"
Universidad Nacional de Jujuy

¿Qué es una pila?

Índice

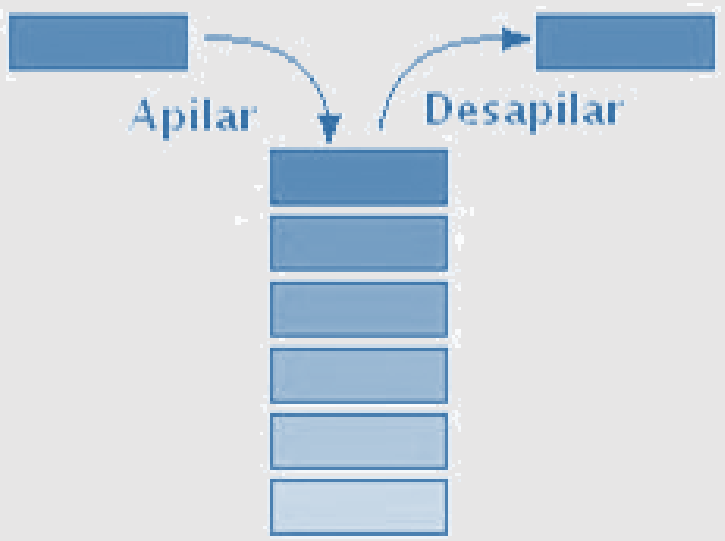
- Definición del TDA pila
- Operaciones fundamentales
- Implementación
- Aplicaciones

Definición (1)

- Una pila es una colección ordenada de elementos, con 3 características:
  - contiene elementos del mismo tipo (estructura homogénea),
  - la recuperación de elementos se realiza en orden inverso al de almacenamiento (acceso **LIFO**) y
  - la cantidad de elementos almacenados varía durante la ejecución (estructura dinámica).

Definición (2)

- En una pila, los elementos se almacenan a partir de una dirección de memoria ocupando posiciones consecutivas y ordenadas.
- La inserción o recuperación de elementos siempre se realiza por el tope o cima de la pila.



Operaciones Fundamentales (1)

- Sobre el TDA pila se definen las siguientes operaciones:
  - Iniciar pila (*Init_Stack*)
  - Agregar elemento (*Push_Stack*)
  - Extraer elemento (*Pop_Stack*)
  - Determinar pila vacía (*Empty_Stack*)
  - Determinar pila llena (*Full_Stack*)
  - Consultar último elemento (*Top_Stack*)

Operaciones Fundamentales (2)

- *Init_Stack (inicializar pila)*
  - Propósito: crear una pila vacía.
  - Entrada: pila de datos.
  - Salida: pila de datos vacía.
  - Restricciones: ninguna.

Operaciones Fundamentales (3)

- *Push_Stack (apilar elemento)*
  - Propósito: agregar un elemento a la pila de datos.
  - Entrada: pila de datos y nuevo elemento.
  - Salida: pila de datos con un nuevo elemento en la cima.
  - Restricciones: pila inicializada y contenedor de datos no completo.

Operaciones Fundamentales (4)

- *Full_Stack (pila llena)*
  - Propósito: determinar si el contenedor de datos de la pila está completo.
  - Entrada: pila de datos.
  - Salida: valor lógico *true* si el contenedor está completo o *false* en caso contrario.
  - Restricciones: pila inicializada.

Operaciones Fundamentales (5)

- *Pop_Stack (desapilar elemento)*
  - Propósito: quitar el elemento de la cima de la pila de datos.
  - Entrada: pila de datos.
  - Salida: pila de datos con un elemento menos, se modifica la cima.
  - Restricciones: pila inicializada y contenedor de datos no vacío.

Operaciones Fundamentales (6)

- *Empty_Stack (pila vacía)*
  - Propósito: determinar si el contenedor de datos de la pila está vacío.
  - Entrada: pila de datos.
  - Salida: valor lógico *true* si el contenedor está vacío o *false* en caso contrario.
  - Restricciones: pila inicializada.

Operaciones Fundamentales (7)

- *Top_Stack (tope de la pila)*
  - Propósito: consultar el elemento de la cima pila de datos.
  - Entrada: pila de datos.
  - Salida: elemento de la cima de la pila de datos.
  - Restricciones: pila inicializada.

Implementación: Arreglos (1)

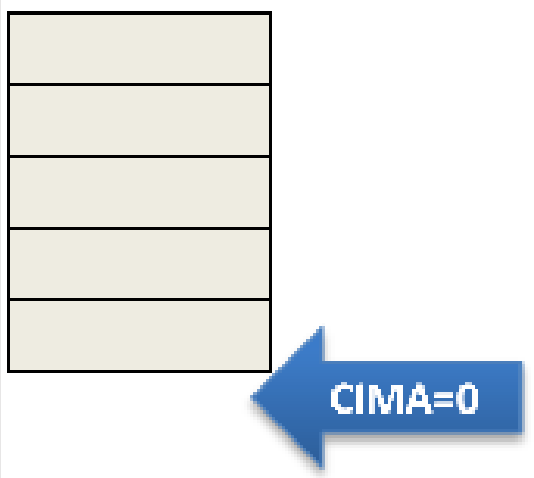
- De acuerdo a la especificación del TDA pila se seleccionan las estructuras de datos y algoritmos que permitan realizar la implementación:
  - Estructura de datos: Se requiere un contenedor de datos y un indicador del último elemento almacenado.

```
contenedor=ARREGLO [1..MAX] de ELEMENTOS
tpila=REGISTRO
        datos:contenedor
        cima:ENTERO
      FIN_REGISTRO
```

Implementación: Arreglos (2)

- Operación *init_stack*

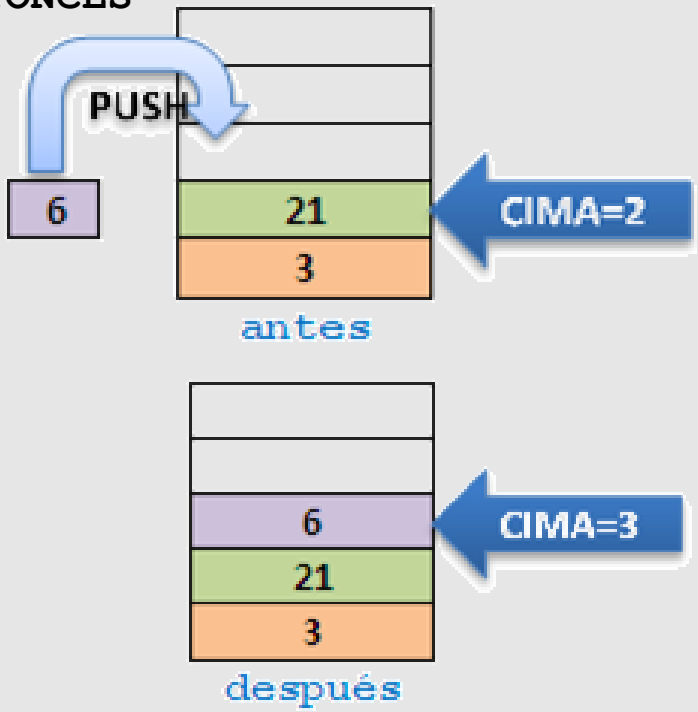
```
PROCEDIMIENTO init_stack(E/S pila: tpila)
INICIO
        pila.cima←0
FIN
```



Implementación: Arreglos (3)

- Operación *push_stack*

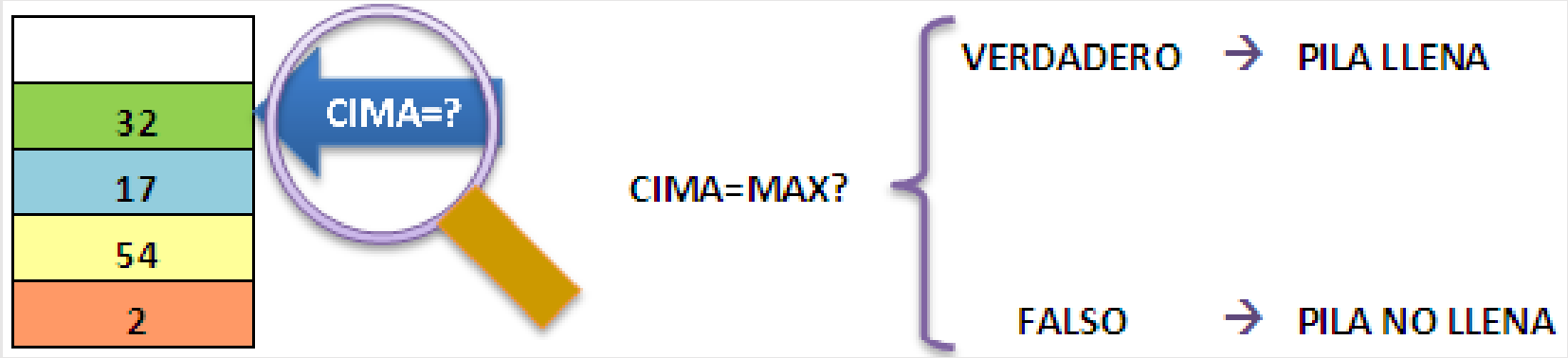
```
PROCEDIMIENTO push_stack(E/S pila:tpila,E nuevo:ELEMENTO)
INICIO
      SI full_stack(pila)=VERDADERO ENTONCES
           ESCRIBIR "Pila llena"
      SINO
           pila.cima←pila.cima+1
           pila.datos[pila.cima]←nuevo
      FIN_SI
FIN
```



Implementación: Arreglos (4)

- Operación *full_stack*

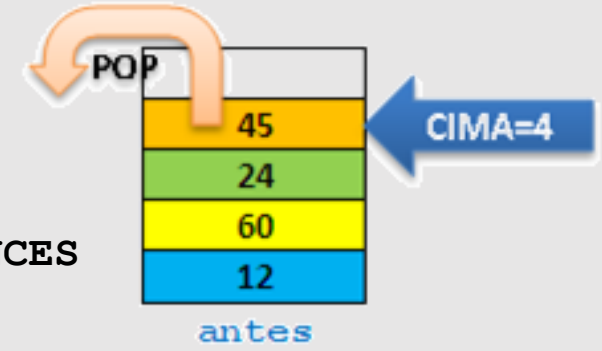
```
FUNCIÓN full_stack(E pila:tpila): LÓGICO
INICIO
        full_stack ← pila.cima=MAX
FIN
```

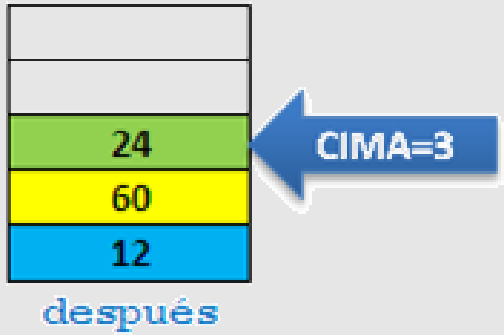


Implementación: Arreglos (5)

- Operación *pop_stack*

```
FUNCIÓN pop_stack(E/S pila:tpila): ELEMENTO
VARIABLES
      extraido: ELEMENTO
INICIO
      SI empty_stack(pila)=VERDADERO ENTONCES
           extraido←valor arbitrario
      SINO
           extraido←pila.datos[pila.cima]
           pila.cima←pila.cima-1
      FIN_SI
      pop_stack←extraido
FIN
```





Implementación: Arreglos (6)

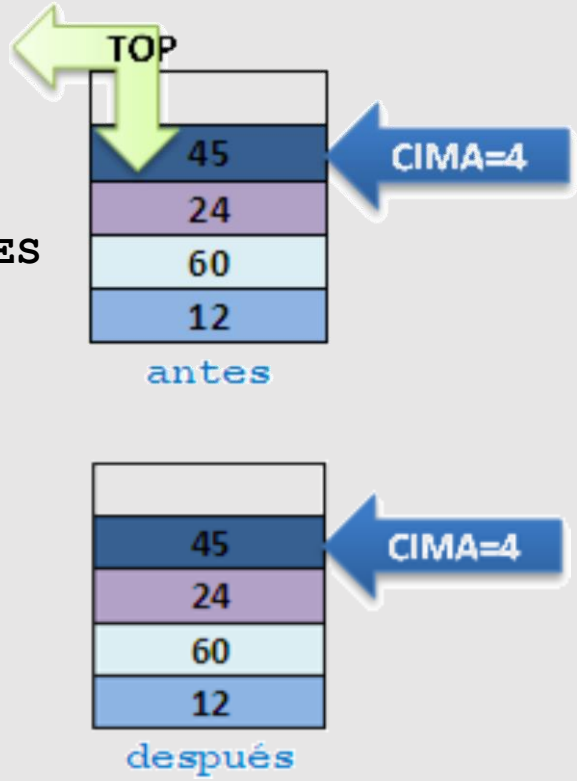
- Operación *empty_stack*

```
FUNCIÓN empty_stack(E pila:tpila): LÓGICO
INICIO
        empty_stack ← pila.cima=0
FIN
```

Implementación: Arreglos (7)

- Operación *top_stack*

```
FUNCIÓN top_stack(E pila:tpila): ELEMENTO
VARIABLES
      consultado: ELEMENTO
INICIO
      SI empty_stack(pila)=VERDADERO ENTONCES
           consultado←valor arbitrario
      SINO
           consultado←pila.datos[pila.cima]
      FIN_SI
      top_stack←consultado
FIN
```



Implementación Modificada (1)

- Modifique la implementación básica del **TDA pila** de forma que el **contenedor de datos** y la **cima o tope** de la pila se almacenen en un único arreglo.
- ¿Cómo se modifican las **operaciones** de pila para esta implementación?

Implementación Modificada (2)

- TDA Pila modificado

```
const int MAX=10;
typedef int tpila[MAX];
```

- Operaciones modificadas: *init_stack*

```
void init_stack (tpila &p)
{
     p[MAX-1]←-1;
}
```

Implementación Modificada (3)

- TDA Pila modificado

```
const int MAX=10;
typedef int tpila[MAX];
```

- Operaciones modificadas: *full_stack*

```
bool full_stack (tpila p)
{
     return p[MAX-1]==MAX-2;
}
```

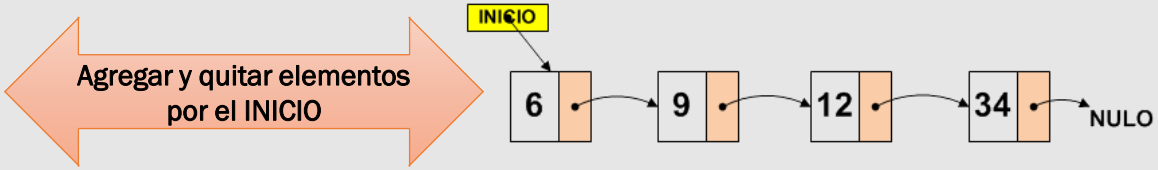
Implementación Modificada (2)

- Operaciones modificadas: *push_stack*

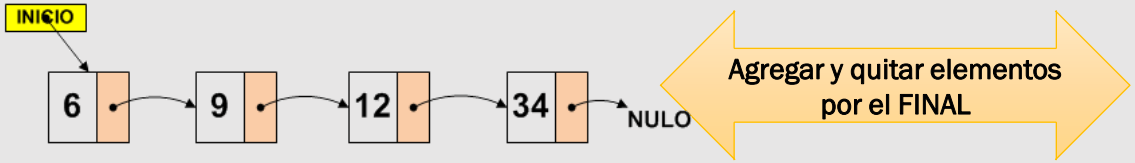
```
void push_stack(tpila p,int nuevo)
{
     if (full_stack(p)==true)
       cout << "PILA LLENA" << endl;
     else
       { p[MAX-1]=p[MAX-1]+1;
         p[p[MAX-1]]=nuevo;
       }
}
```

Implementación: Listas (1)

- Implementación del TDA Pila mediante **listas simples**
  - Alternativa 1:
    - Utilizar las operaciones *agregar_inicio* y *quitar_inicio* para representar el comportamiento de la pila.

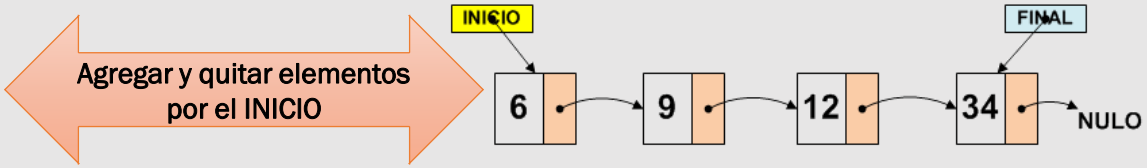


  - Alternativa 2:
    - Utilizar las operaciones *agregar_final* y *quitar_final* para representar el comportamiento de la pila.

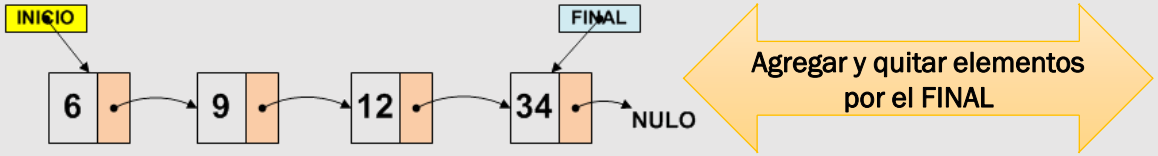


Implementación: Listas (2)

- Implementación del TDA Pila mediante **listas simples**
  - Alternativa 3:
    - Utilizar las operaciones *agregar_inicio* y *quitar_inicio* para representar el comportamiento de la pila.

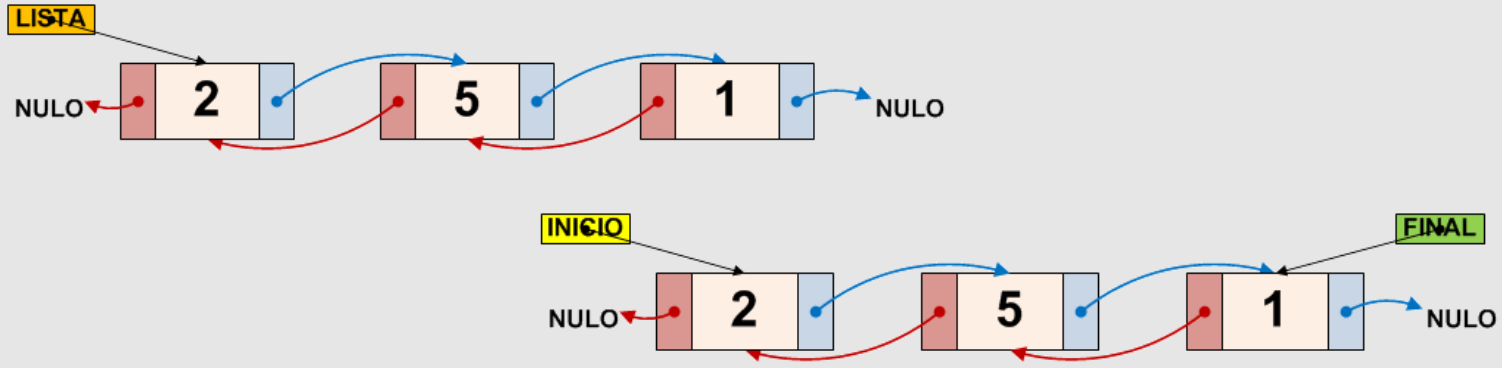


  - Alternativa 4:
    - Utilizar las operaciones *agregar_final* y *quitar_final* para representar el comportamiento de la pila.



Implementación: Listas (3)

- Implementación del TDA Pila mediante **listas dobles**
  - ¿Cuáles son las alternativas de implementación al utilizar listas dobles?



  - ¿Cuáles son las operaciones de listas dobles que pueden utilizarse para implementar las operaciones de pila para cada alternativa?

Aplicaciones

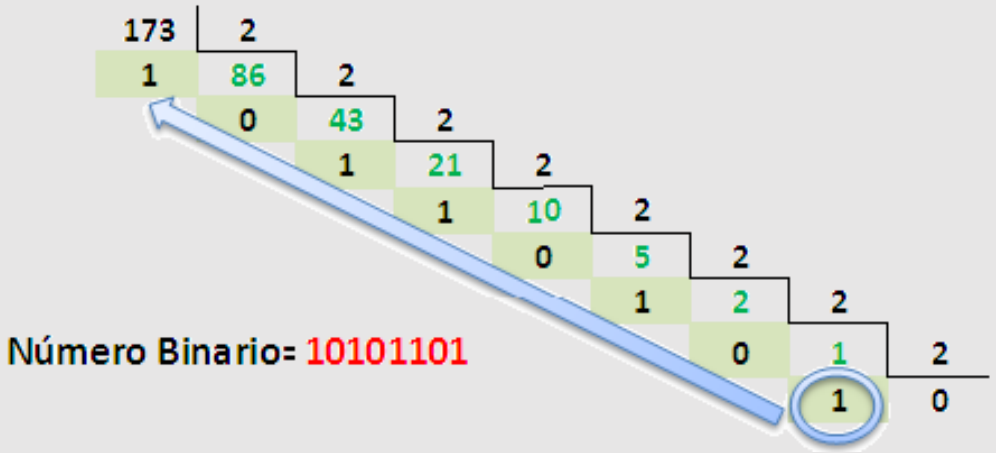
- El concepto de pila puede aplicarse para resolver:
  - inversión de cadenas, detección de palíndromos,
  - verificación de parentización,
  - reconocimiento de lenguaje (análisis sintáctico),
  - conversión de notaciones (por ej., notación interfija a posfija),
  - evaluación de expresiones posfijas,
  - cambio de base (aritmética en base origen)
  - paso de parámetros,
  - recursividad, etc.

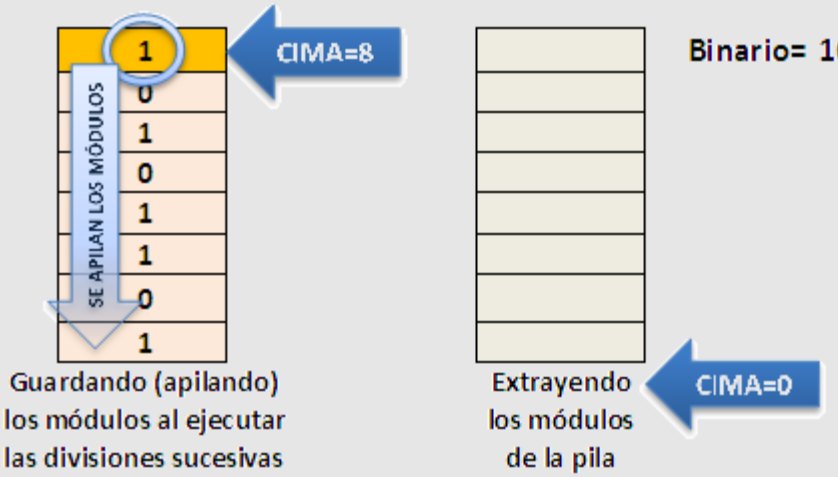
Ejemplo de Aplicación (1)

- Diseñe un algoritmo que realice el cambio de base de un número decimal positivo al sistema binario, aplicando el método aritmética en base origen. Utilice en la solución el TDA pila.
- El cambio de base se realiza dividiendo, sucesivamente, el número por la base 2 hasta que el dividendo sea cero.
- Por cada cociente debe almacenarse el resto obtenido, luego éstos constituirán el número en la base destino (se disponen en orden inverso al que fueron generados).

Ejemplo de Aplicación (2)

- El método de cambio de base puede replicarse mediante un algoritmo que, utilizando pilas, almacene los restos o módulos de la división sucesiva.
- Luego, al extraer los módulos de la pila, es posible formar el número binario correspondiente.





Ejemplo de Aplicación (3)

- El siguiente procedimiento realiza el cambio de base, aplicando el concepto de pila.

```
PROCEDIMIENTO binario(E numero:entero,E destino:entero)
VARIABLES
      p:tpila
INICIO
      init_stack(p)
      MIENTRAS (numero<>0) HACER
            push_stack(p,numero mod destino)
            numero←numero div destino
      FIN_MIENTRAS
      MIENTRAS (empty_stack(p)<> VERDADERO) HACER
            ESCRIBIR pop_stack(p)
      FIN_MIENTRAS
FIN
```

Ejemplo de Aplicación (4)

- Codificación en C/C++ del procedimiento de cambio de base.

```
void binario(int n, int d)
{ tpila pila;
  iniciar_pila(pila);
  while (n!=0)
  { agregar_pila(pila,n%d);
    n=n/d;
  }
  while (pila_vacia(pila)!=true)
    cout << extraer_pila(pila);
  cout << endl;
}
```

Ejemplo de Aplicación (5)

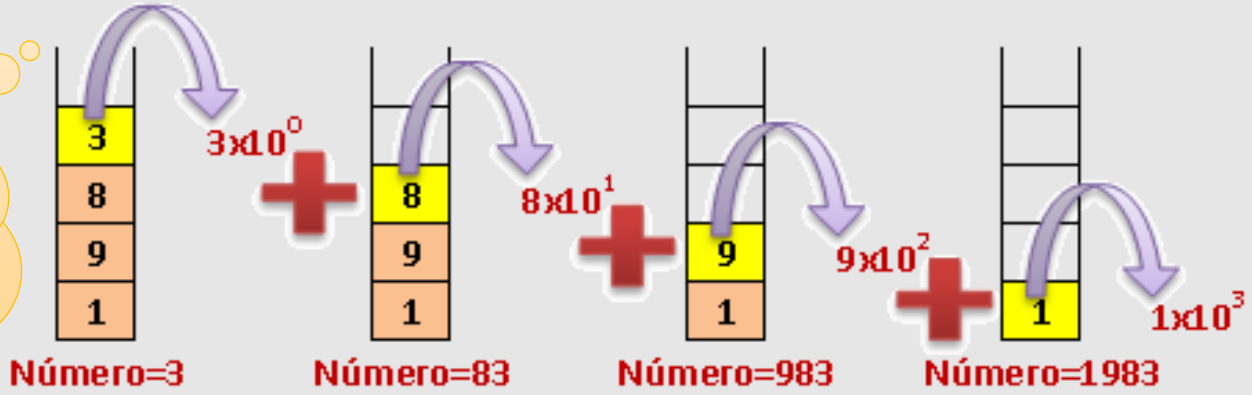
- Utilizando el TDA pila (implementado con listas simples) y sus operaciones básicas, diseñe un algoritmo que convierta los dígitos almacenados en una cadena de caracteres a su correspondiente valor entero.
- Tenga en cuenta que un número entero puede expresarse como una ecuación polinómica, por ejemplo, el valor 235 es igual a

$$2 \times 10^2 + 3 \times 10^1 + 5 \times 10^0$$

Ejemplo de Aplicación (6)

Se recorre la cadena de entrada almacenando cada dígito en la pila de dígitos.

Se extrae cada dígito, y se calcula el término correspondiente de la ecuación polinómica.



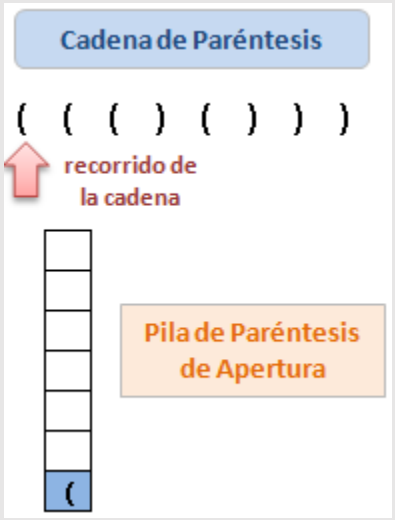
Ejemplo de Aplicación (7)

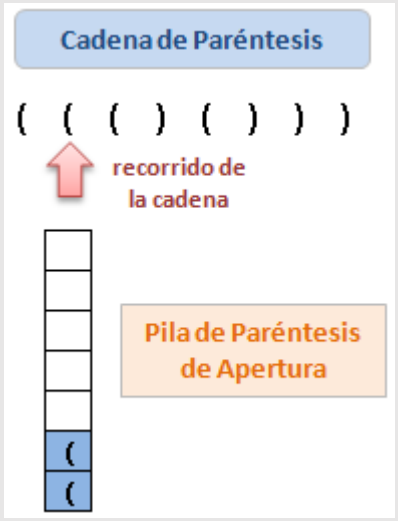
- Conversión de cadena a valor numérico

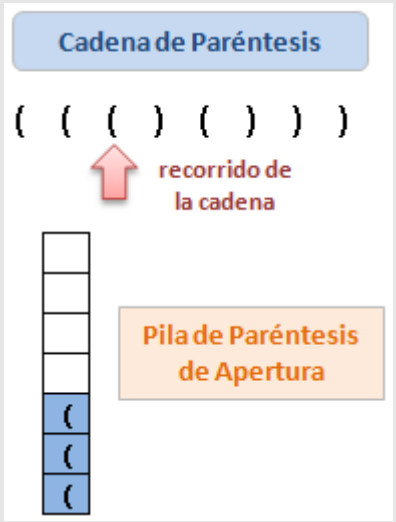
```
int convertir_numero(tcad entrada)
{ pnodo listapila,nuevo,extraido;
  int i;
  float num=0;
  inicia_lista(listapila);
  for(i=0;i<strlen(entrada);i++)
   { crear_nodo(nuevo,entrada[i]-48);
     if (nuevo!=NULL)
       agregar_inicio(listapila,nuevo); }
  for(i=0;listapila!=NULL;i++)
   { extraido=quitar_inicio(listapila);
     num=num+extraido->dato*pow(10.0,i);
     delete(extraido); }
  return num;
}
```

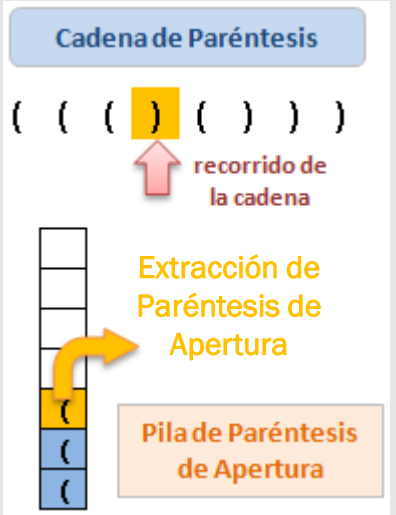
Ejemplo de Aplicación (8)

- Verificación de Parentización



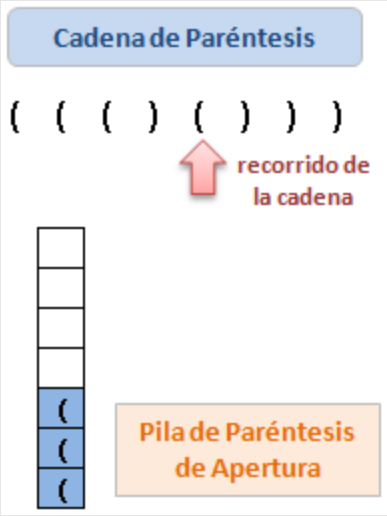


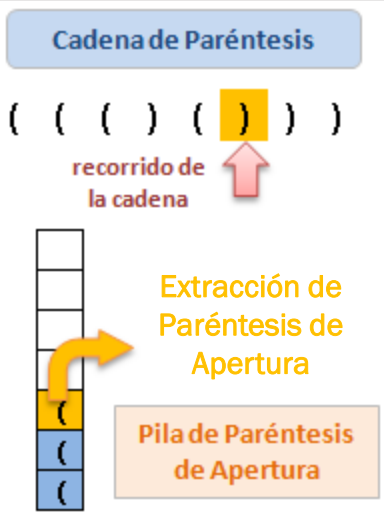


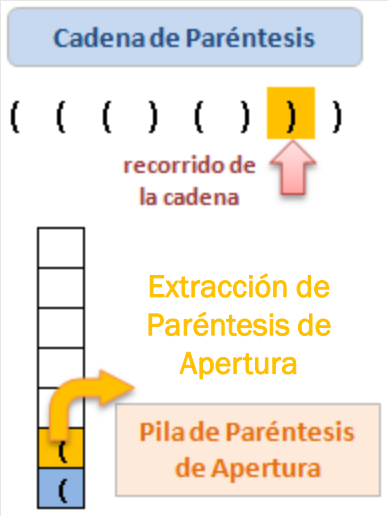


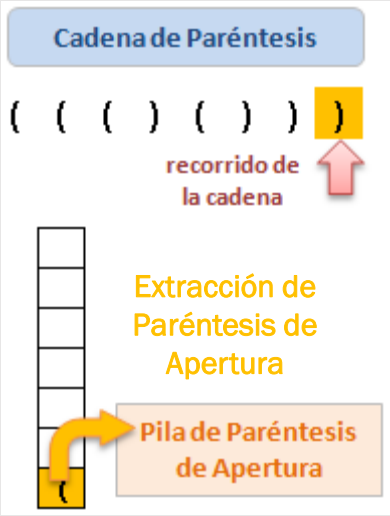
Ejemplo de Aplicación (9)

- Verificación de Parentización



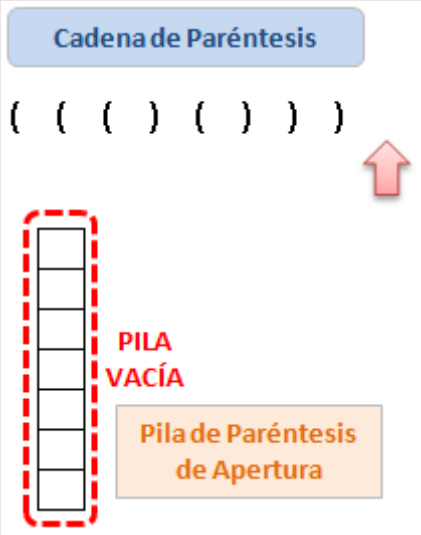






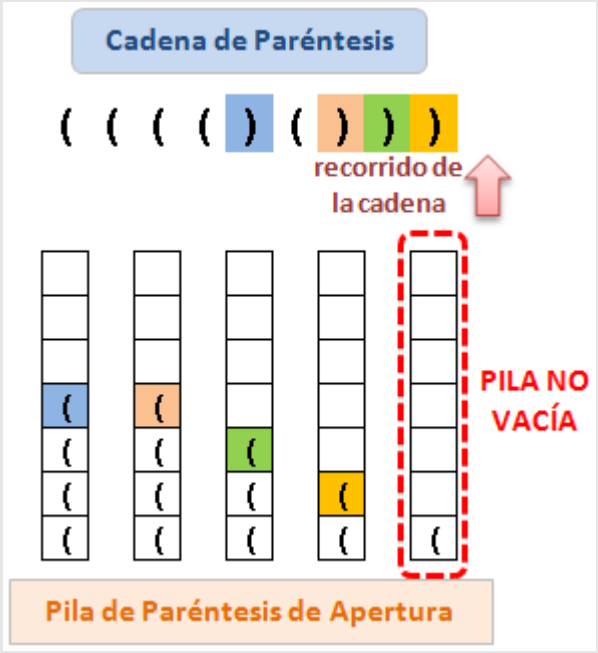
Ejemplo de Aplicación (10)

- Verificación de Parentización

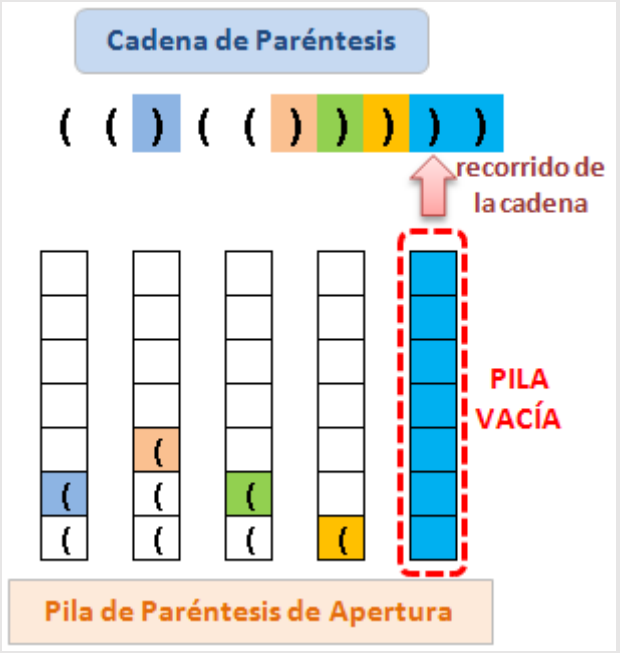


Ejemplo de Aplicación (11)

- Verificación de Parentización. Casos de Error



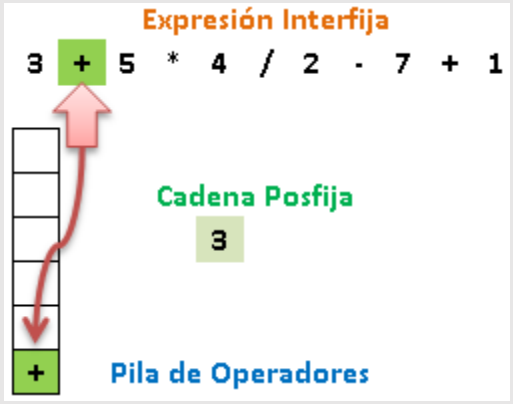
La cadena de entrada contiene más paréntesis de apertura que de cierre.

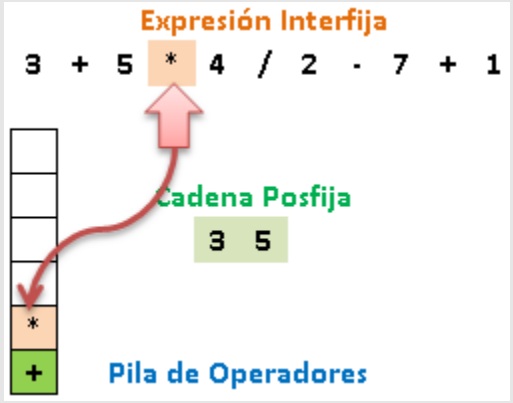


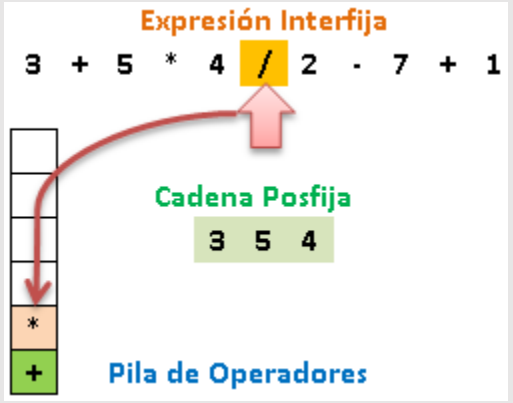
La cadena de entrada contiene más paréntesis de cierre que de apertura, o están dispuestos incorrectamente.

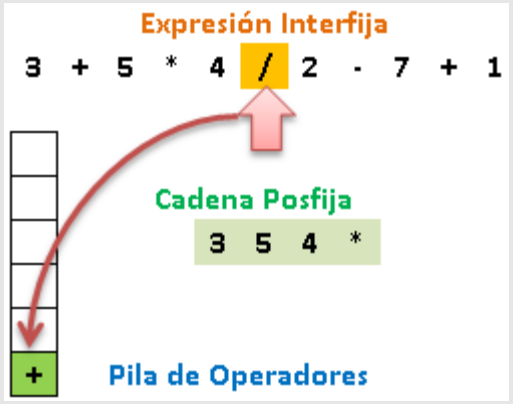
Ejemplo de Aplicación (12)

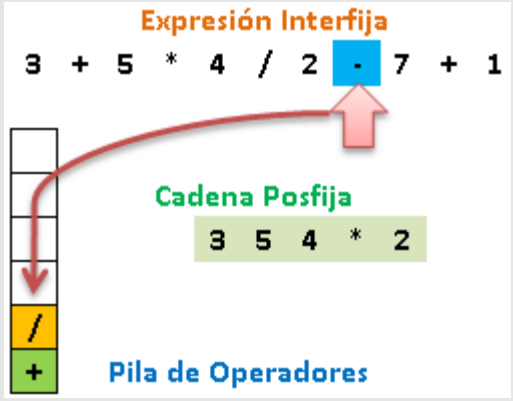
- Conversión de expresiones interfijas a posfijas

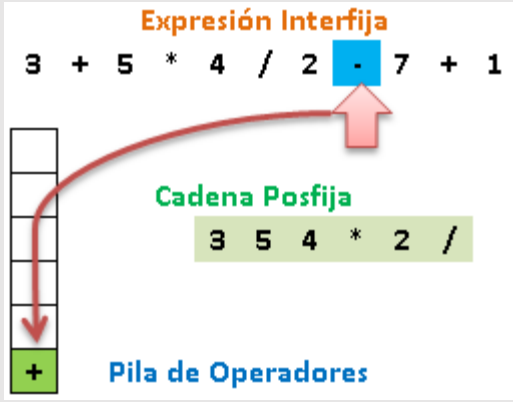






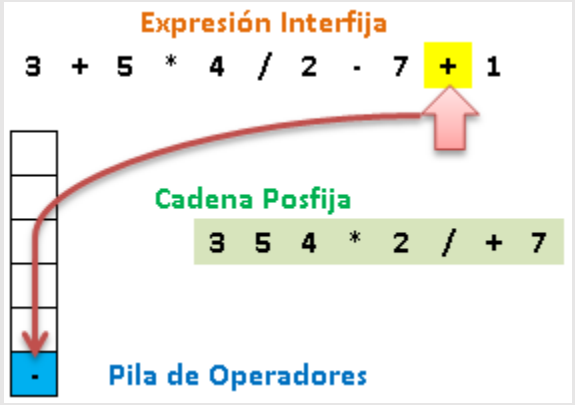


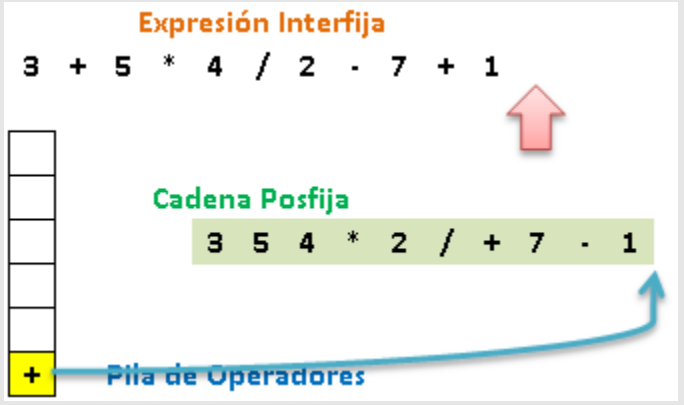


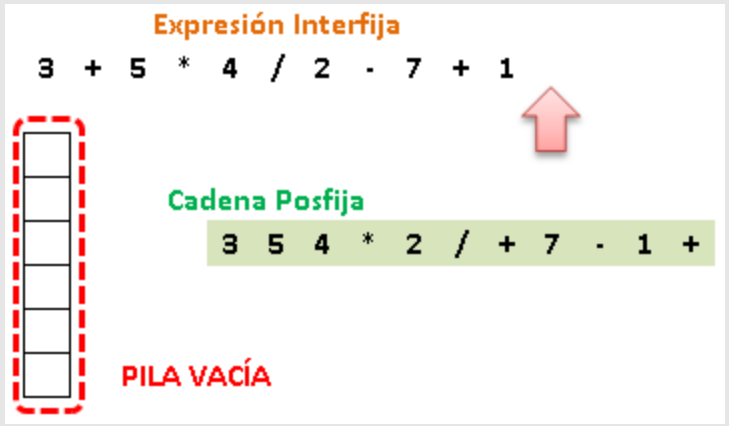


Ejemplo de Aplicación (13)

- Conversión de expresiones interfijas a posfijas

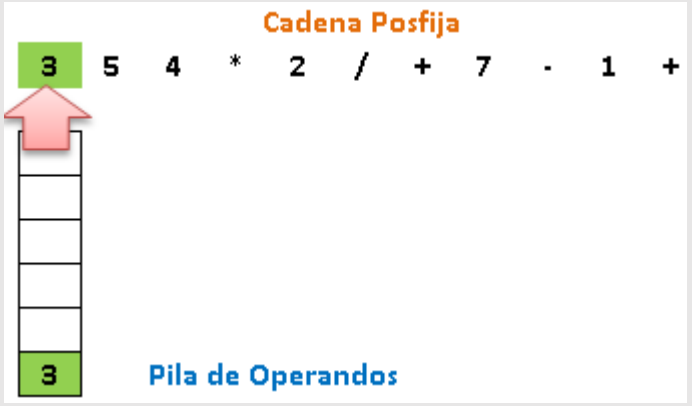


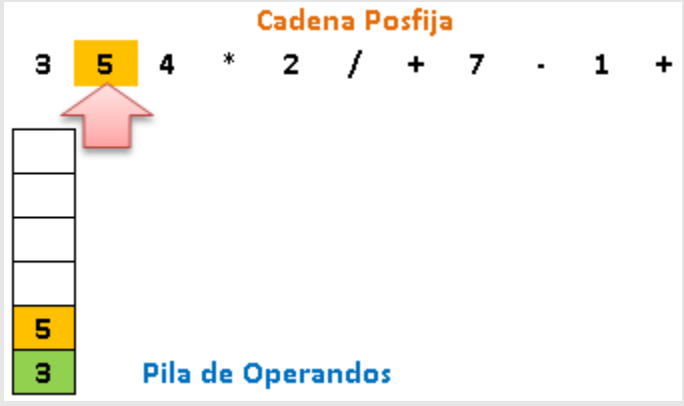




Ejemplo de Aplicación (14)

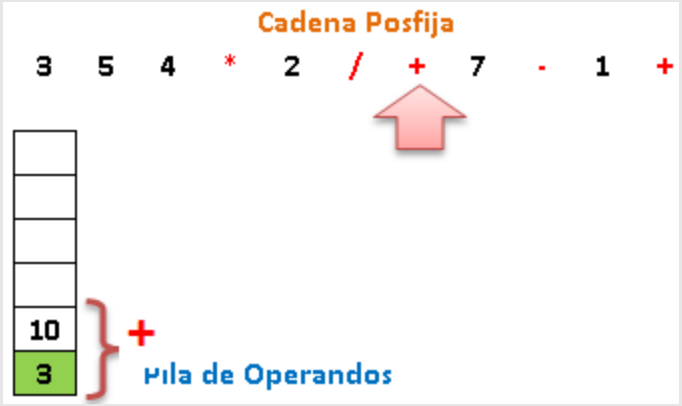
- Cálculo de expresiones posfijas





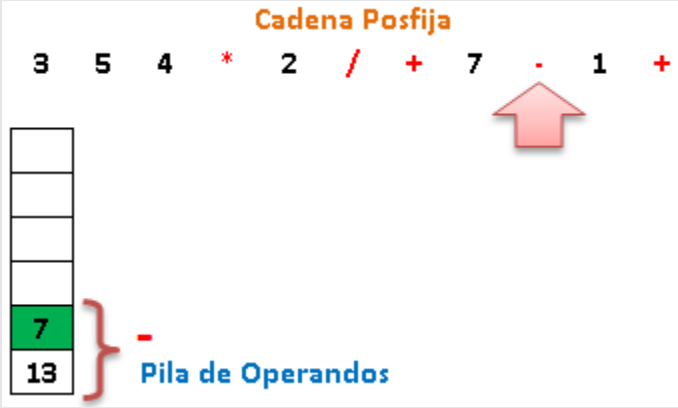
Ejemplo de Aplicación (15)

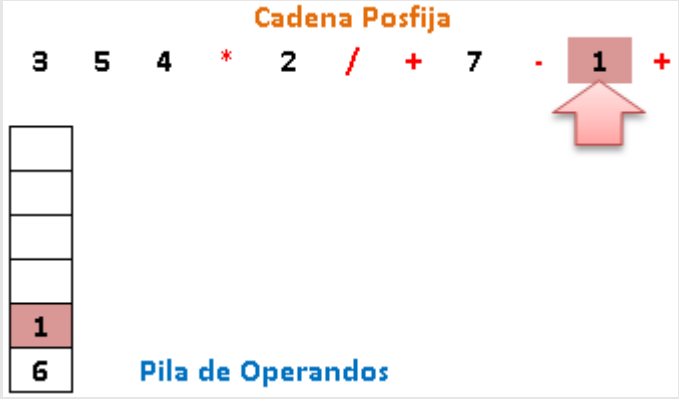
- Cálculo de expresiones posfijas

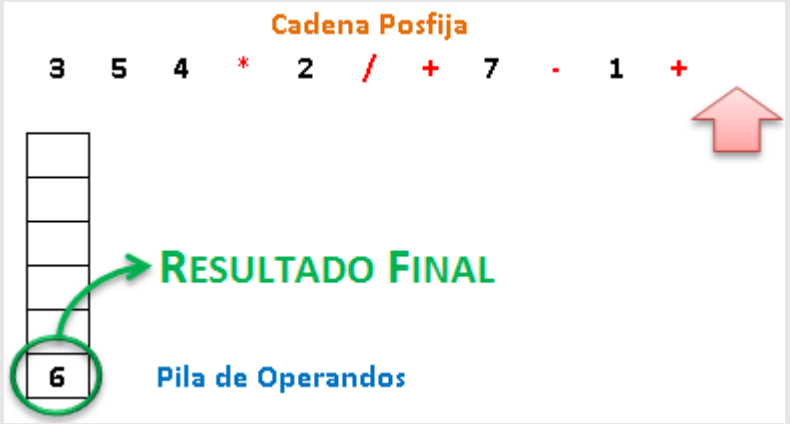


Ejemplo de Aplicación (16)

- Cálculo de expresiones posfijas







Bibliografía

- Joyanes Aguilar *et al*. Estructuras de Datos en C++. Mc Graw Hill. 2007.
- De Giusti, Armando *et al.* Algoritmos, datos y programas, conceptos básicos. Editorial Exacta. 1998.
- Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.