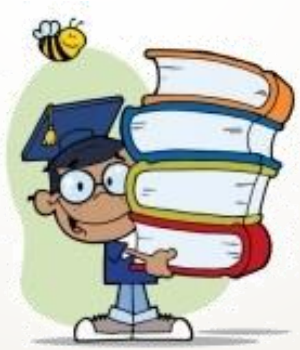
Analista Programador Universitario

Estructura de Datos

Unidad III: TDA Pila



Facultad de Ingeniería
Universidad Nacional de Jujuy

¿Qué es una pila?

Índice

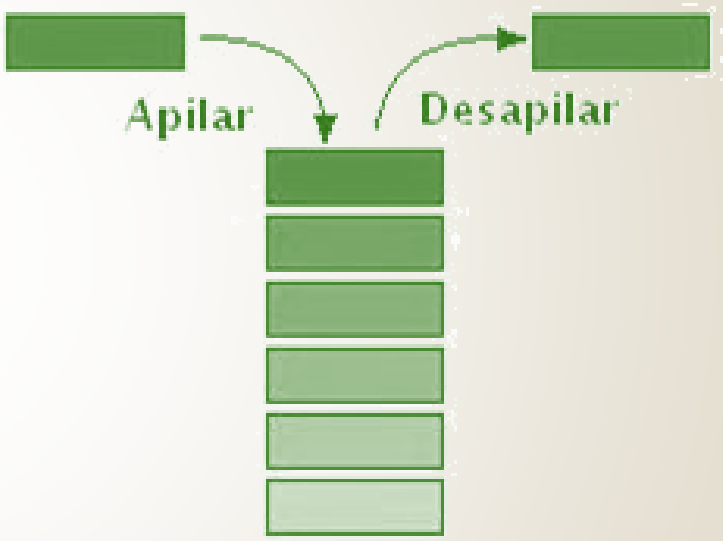
- Definición del TDA pila
- Operaciones fundamentales
- Implementación
- Aplicaciones

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 informática, una pila refiere a un conjunto de posiciones de memoria (consecutivas o no) que se utilizan para almacenar datos.
- La inserción o recuperación de elementos de la pila siempre se realiza por el tope o cima de ésta.



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

Operaciones Fundamentales (3)

- *Push_Stack (apilar o agregar 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 o quitar 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)

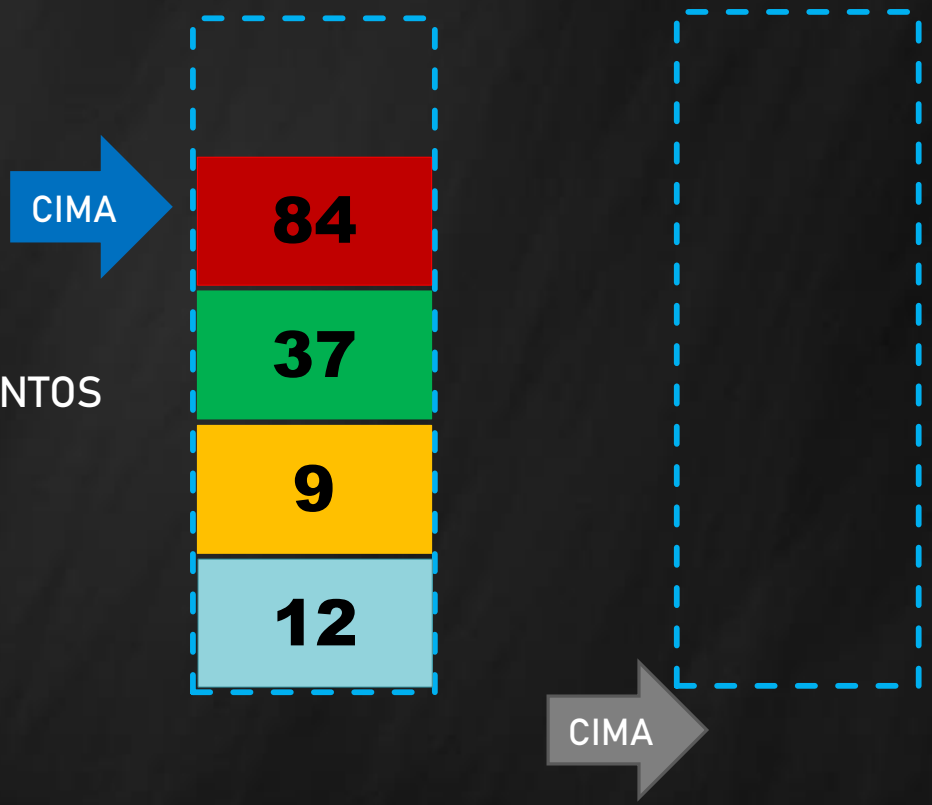
- De acuerdo a la especificación del TDA pila se seleccionan las estructuras de datos y algoritmos que permitan realizar la implementación:
  - Estructura de datos: Se requiere un contenedor de datos y un indicador del último elemento almacenado.

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

Operación Iniciar Pila

Permite crear una pila vacía.

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

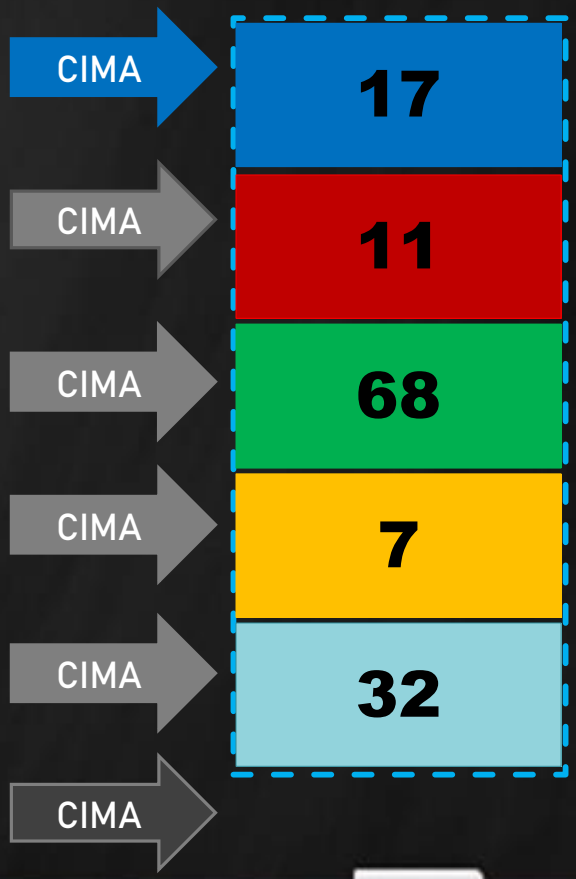



```
contenedor=ARREGLO [1..MAX] de ELEMENTOS
tpila=REGISTRO
        datos:contenedor
        cima:ENTERO
FIN_REGISTRO
```

Operación Agregar Pila

Permite agregar un elemento a una pila, siempre y cuando exista espacio.

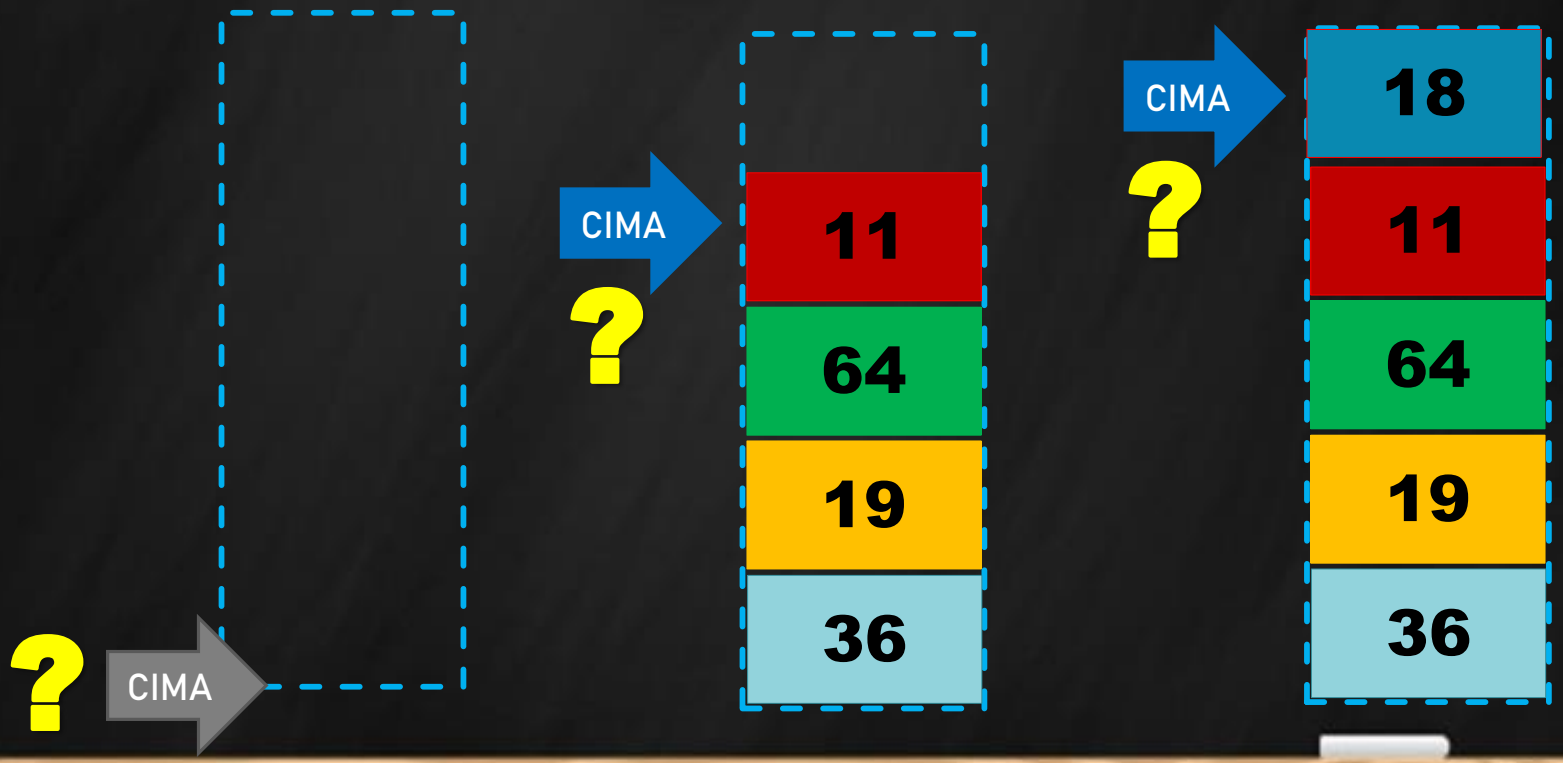
```
PROCEDIMIENTO agregar_pila (E/S pila:tpila, E nuevo:ELEMENTO)
INICIO
    SI pila_llena(pila)=VERDADERO ENTONCES
        ESCRIBIR "Pila llena"
    SINO
        pila.cima←pila.cima+1
        pila.datos[pila.cima]←nuevo
    FIN_SI
FIN
```



Operación Pila Llena

Permite determinar si una pila está llena o no.

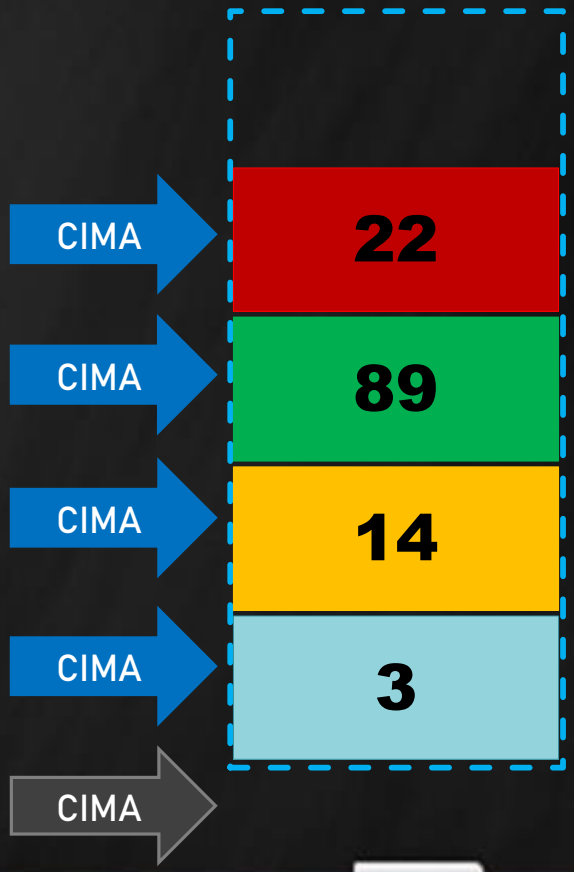
```
FUNCIÓN pila_llena (E pila: tpila): LÓGICO
INICIO
        pila_llena←pila.cima=MAX_PILA
FIN
```



Operación Quitar Pila

Permite quitar un elemento de la cima de una pila, siempre y cuando ésta tenga contenido.

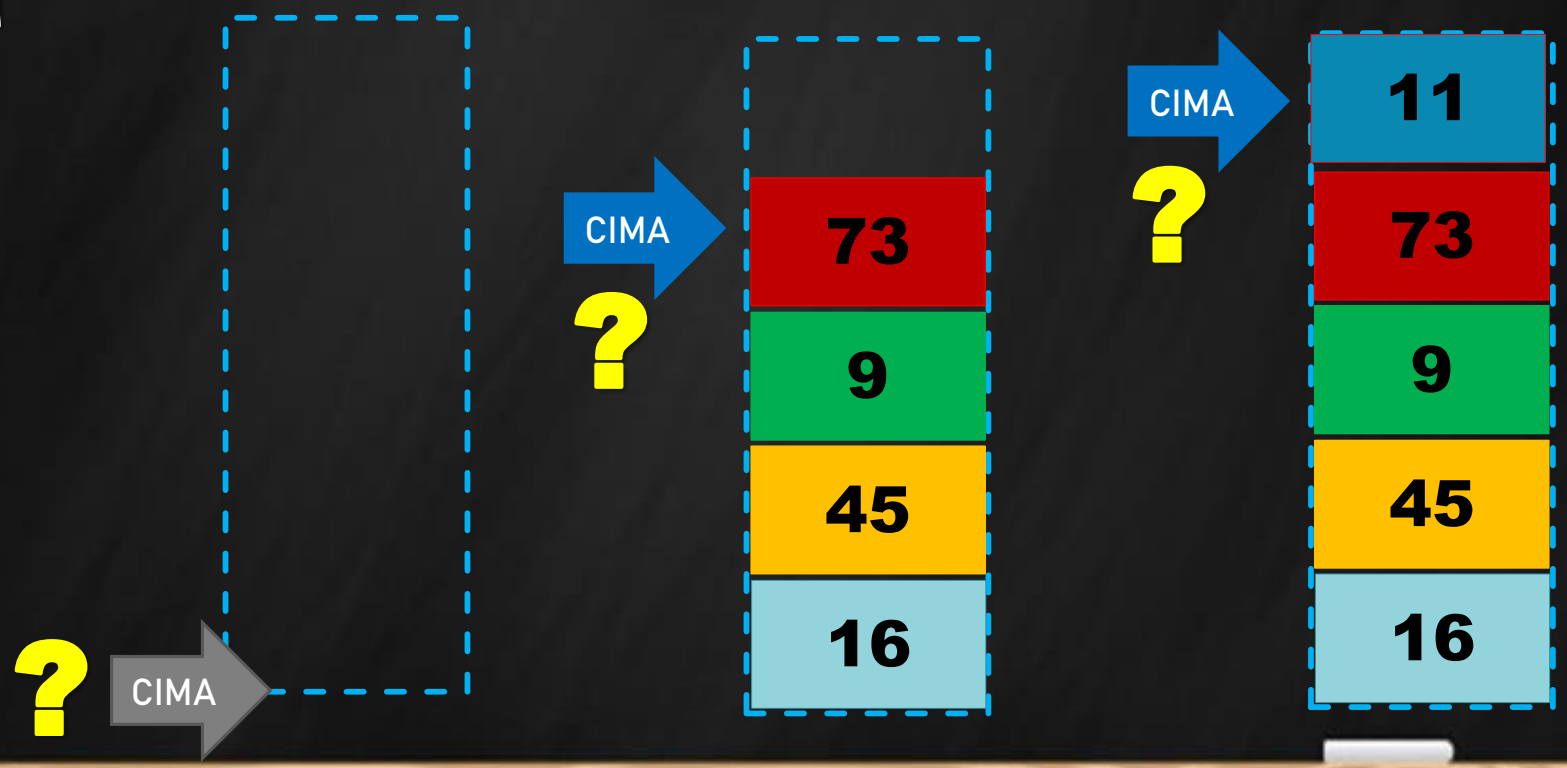
```
FUNCIÓN quitar_pila (E/S pila:tpila): ELEMENTO
VARIABLES
    extraido: ELEMENTO
INICIO
  SI pila_vacia(pila)=VERDADERO ENTONCES
      extraido←valor arbitrario
  SINO
      extraido← pila.datos[pila.cima]
      pila.cima← pila.cima-1
  FIN_SI
  quitar_pila←extraido
FIN
```



Operación Pila Vacía

Permite determinar si una pila está vacía o no.

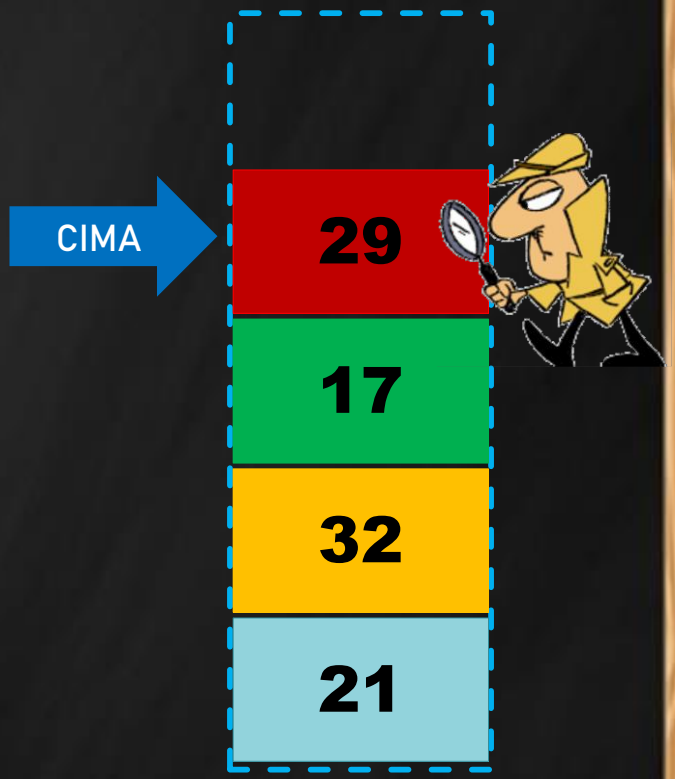
```
FUNCIÓN pila_vacía (E pila: tpila): LÓGICO
INICIO
      pila_vacía←pila.cima=0
FIN
```



Operación Tope Pila

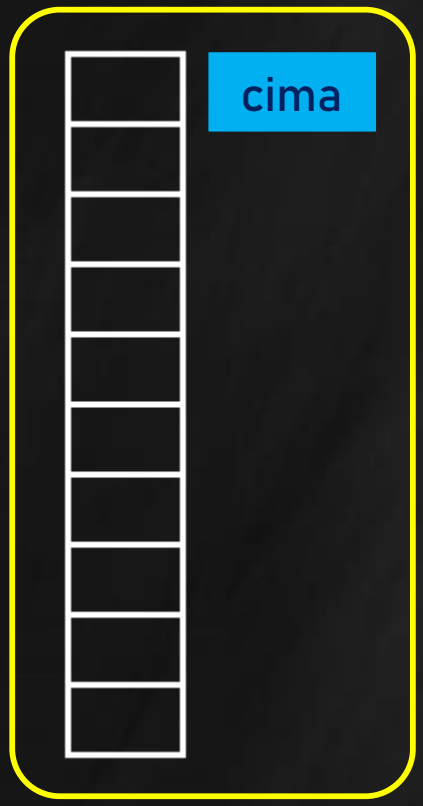
Permite consultar el elemento que se encuentra en la cima de una pila, siempre y cuando ésta tenga contenido.

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



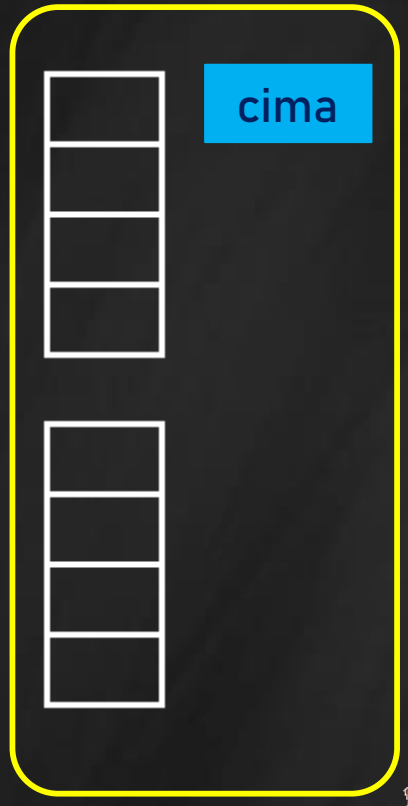
Variantes de Implementación con Arreglos

Básica



1 registro
1 arreglo
1 índice

Variante 1

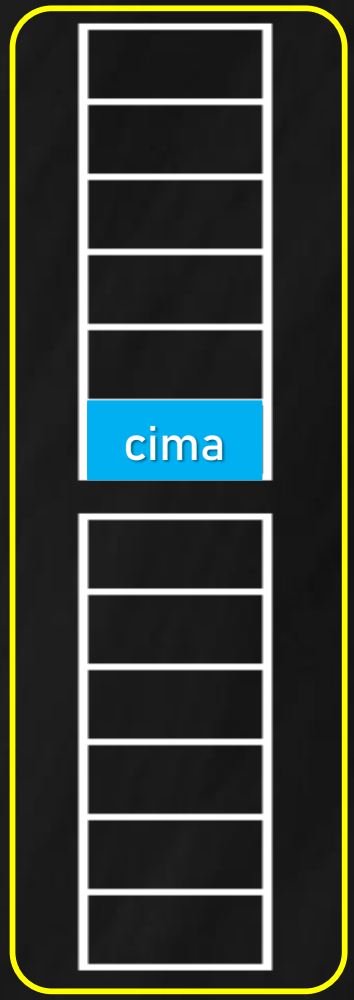


1 registro
2 arreglos
1 índice

Variante 2

1 arreglo

Variante 3



1 registro y 2 arreglos

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: *iniciar_pila*

```
void iniciar_pila (tpila &p)
{
     p[MAX-1]=-1;
}
```

Implementación Modificada (3)

- TDA Pila modificado

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

- Operaciones modificadas: *pila_llena*

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

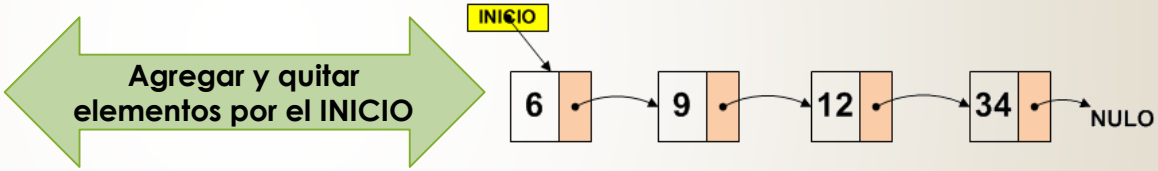
Implementación Modificada (2)

- Operaciones modificadas: *agregar_pila*

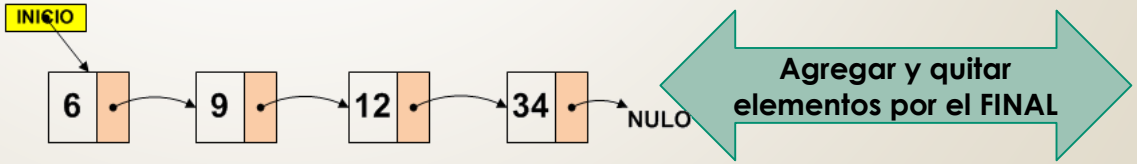
```
void agregar_pila(tpila &p,int nuevo)
{
      if (pila_llena(p)==true)
        cout << "PILA LLENA" << endl;
      else
        { p[MAX-1]=p[MAX-1]+1;
          p[p[MAX-1]]=nuevo;
        }
}
```

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.

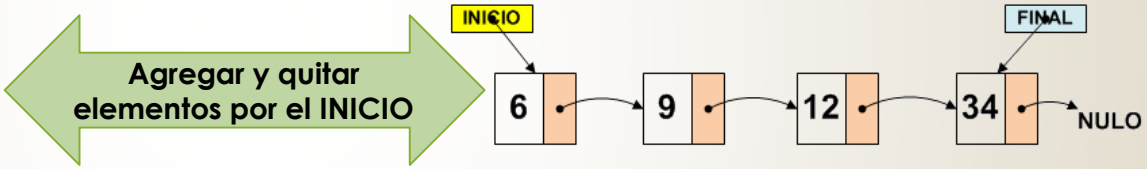


  - Alternativa 2:
    - Utilizar las operaciones *agregar_final* y *quitar_final* para representar el comportamiento de la pila.

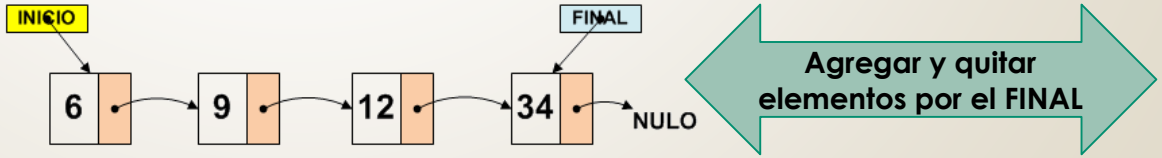


Implementación: Listas (2)

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

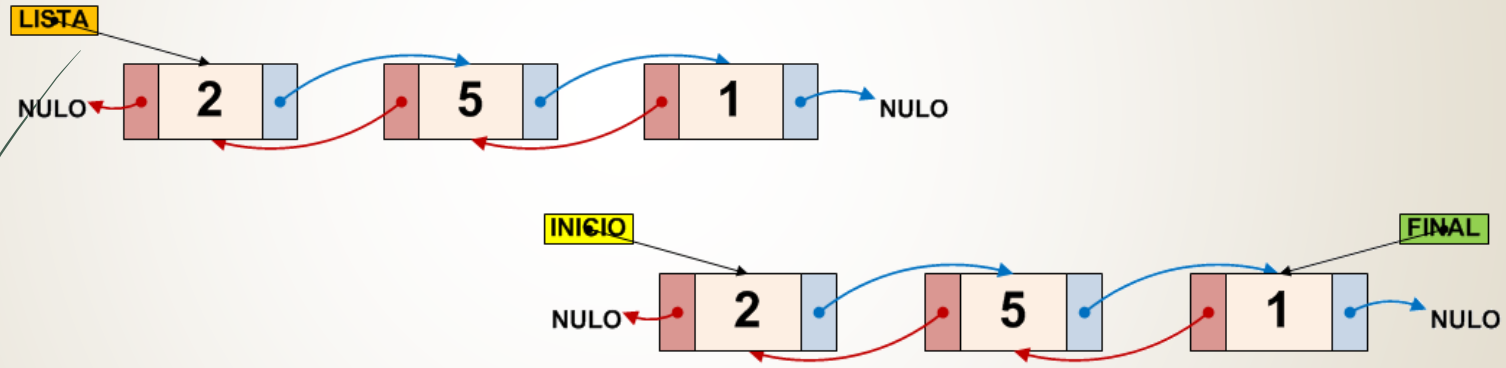


  - Alternativa 4:
    - Utilizar las operaciones *agregar_final* y *quitar_final* para representar el comportamiento de la pila.



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

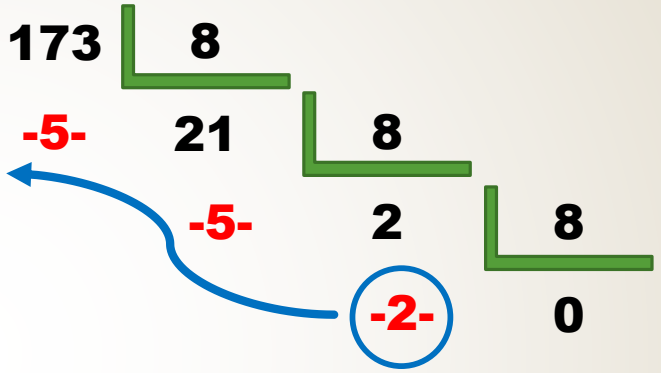
- El concepto de pila puede aplicarse para resolver:
  - inversión de cadenas, detección de palíndromos,
  - verificación de parentización,
  - reconocimiento de lenguaje (análisis sintáctico),
  - conversión de notaciones (por ej., notación interfija a posfija),
  - evaluación de expresiones posfijas,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

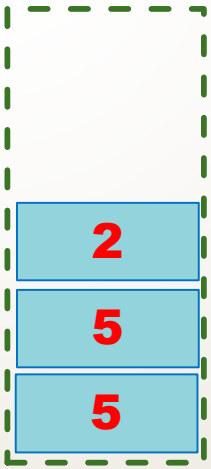
Ejemplo de Aplicación (1)

- Diseñe un algoritmo que realice el cambio de base de un número decimal positivo al sistema octal, 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 8 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 valor octal correspondiente.





255

Ejemplo de Aplicación (3)

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

```
FUNCIÓN octal(E numero:entero):entero
VARIABLES
      p:tpila
      valor:entero
INICIO
      iniciar_pila(p)
      MIENTRAS (numero<>0) HACER
            agregar_pila(p,numero mod 8)
            numero←numero div 8
      FIN_MIENTRAS
      valor←0
      MIENTRAS (pila_vacia(p)<> VERDADERO) HACER
            valor←valor*10+quitar_pila(p)
      FIN_MIENTRAS
      octal←valor
FIN
```

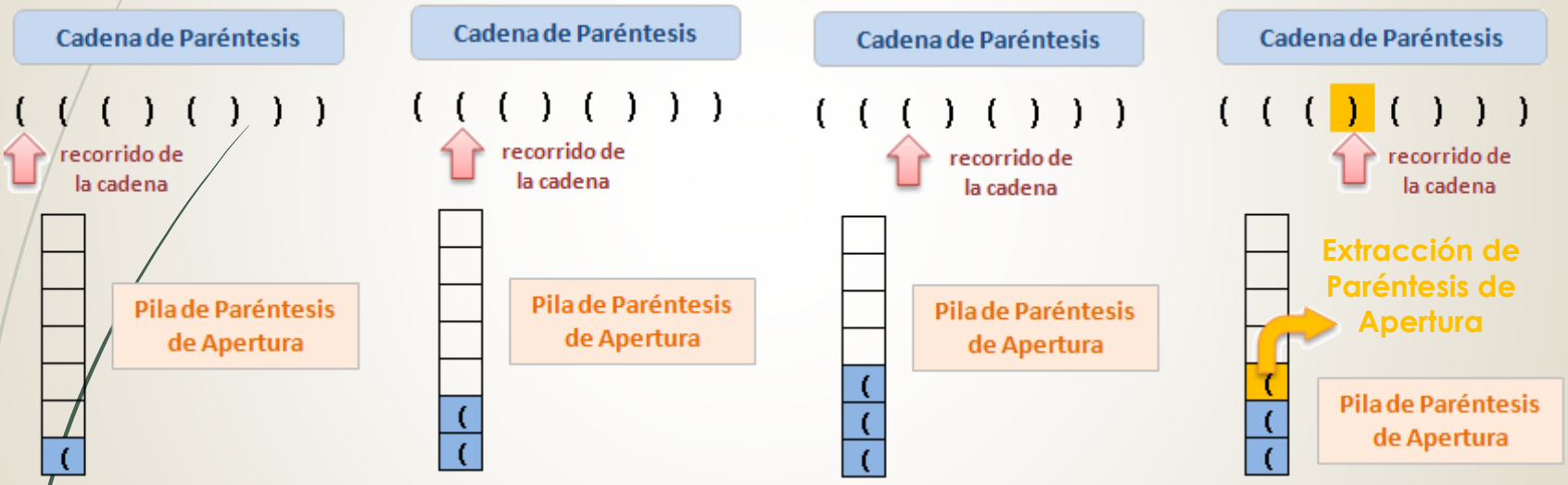
Ejemplo de Aplicación (4)

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

```
int octal(int n)
{ tpila pila;
  int valor=0;
  iniciar_pila(pila);
  while (n!=0)
  { agregar_pila(pila,n%8);
    n=n/8;
  }
  while (pila_vacia(pila)!=true)
     valor=valor*10 + extraer_pila(pila);
  return valor;
}
```

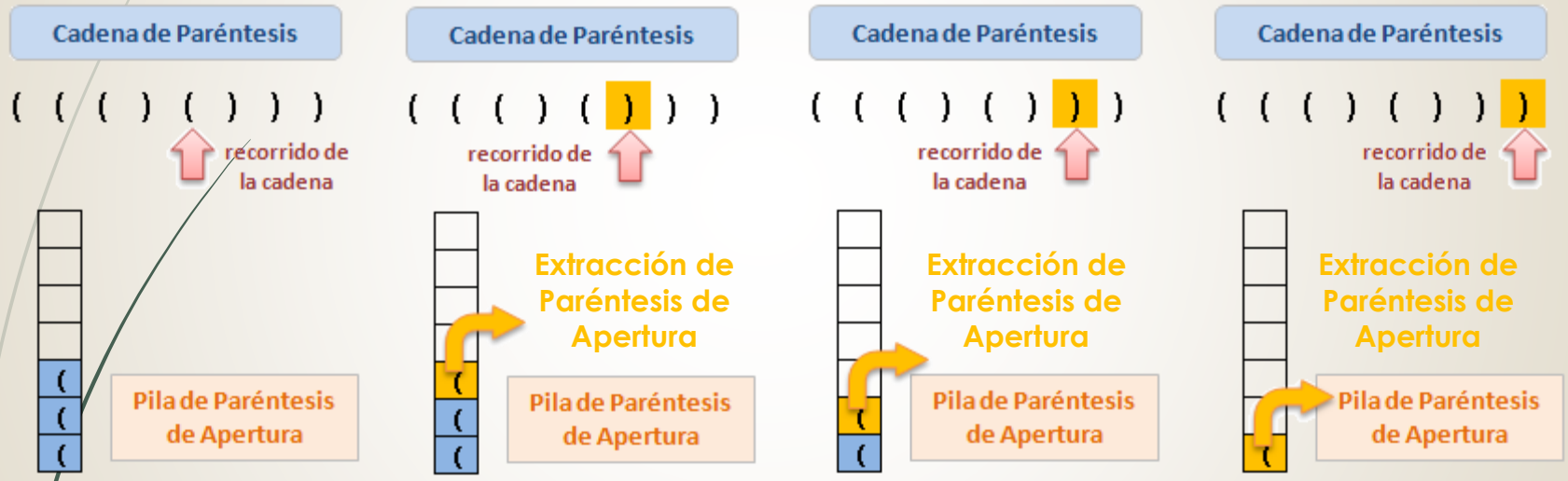
Ejemplo de Aplicación (8)

- Verificación de Parentización



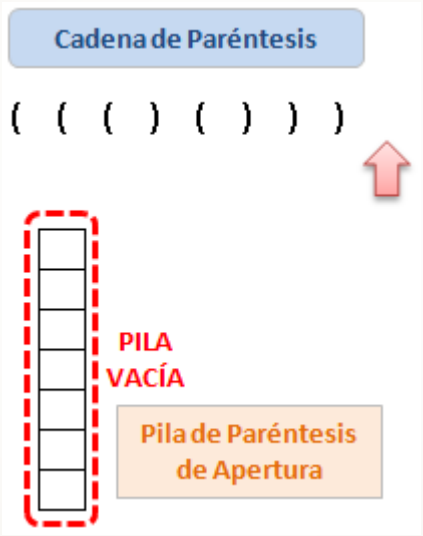
Ejemplo de Aplicación (9)

- Verificación de Parentización



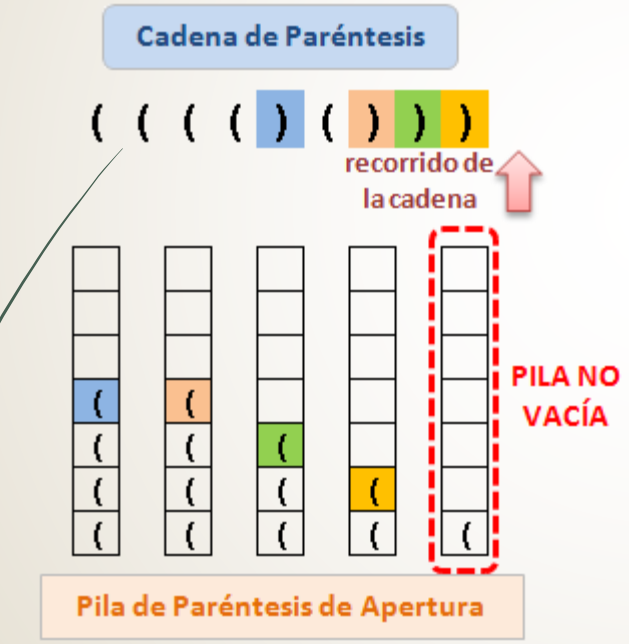
Ejemplo de Aplicación (10)

- Verificación de Parentización

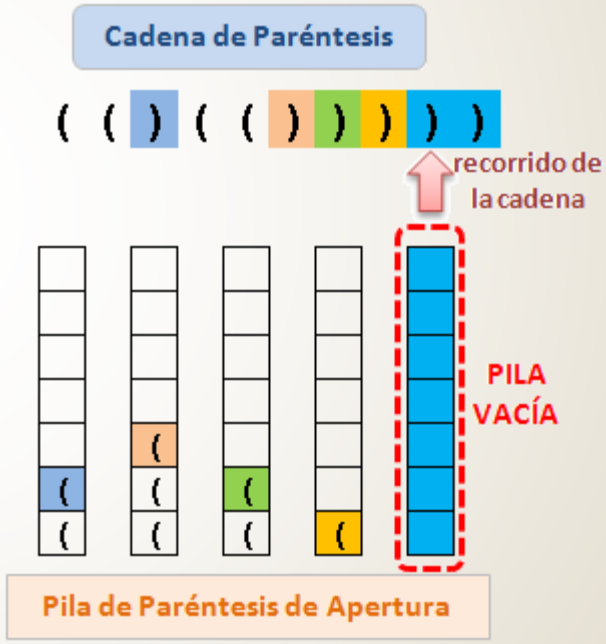


Ejemplo de Aplicación (11)

- Verificación de Parentización. Casos de Error



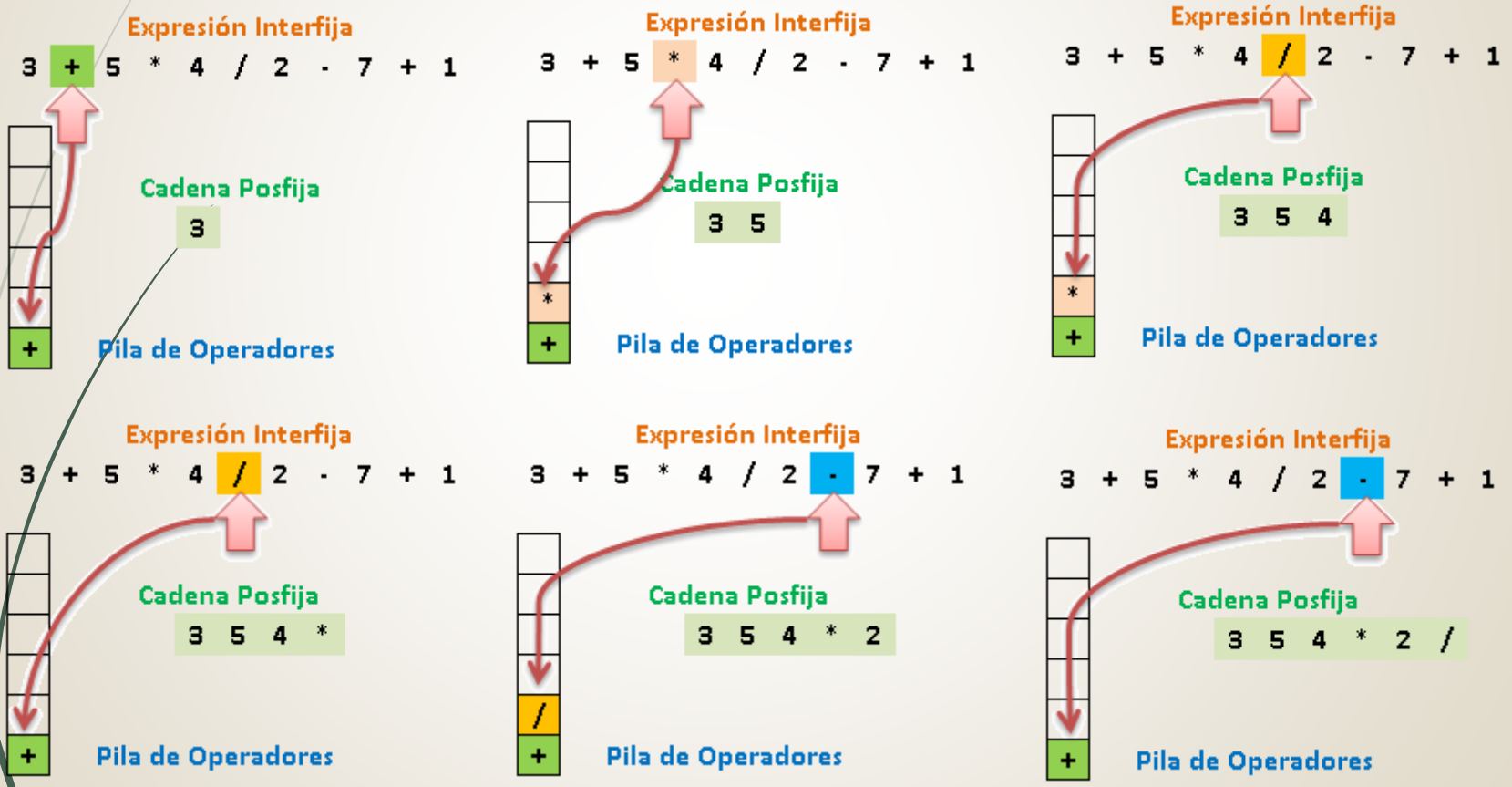
La cadena de entrada contiene más paréntesis de apertura que de cierre.



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

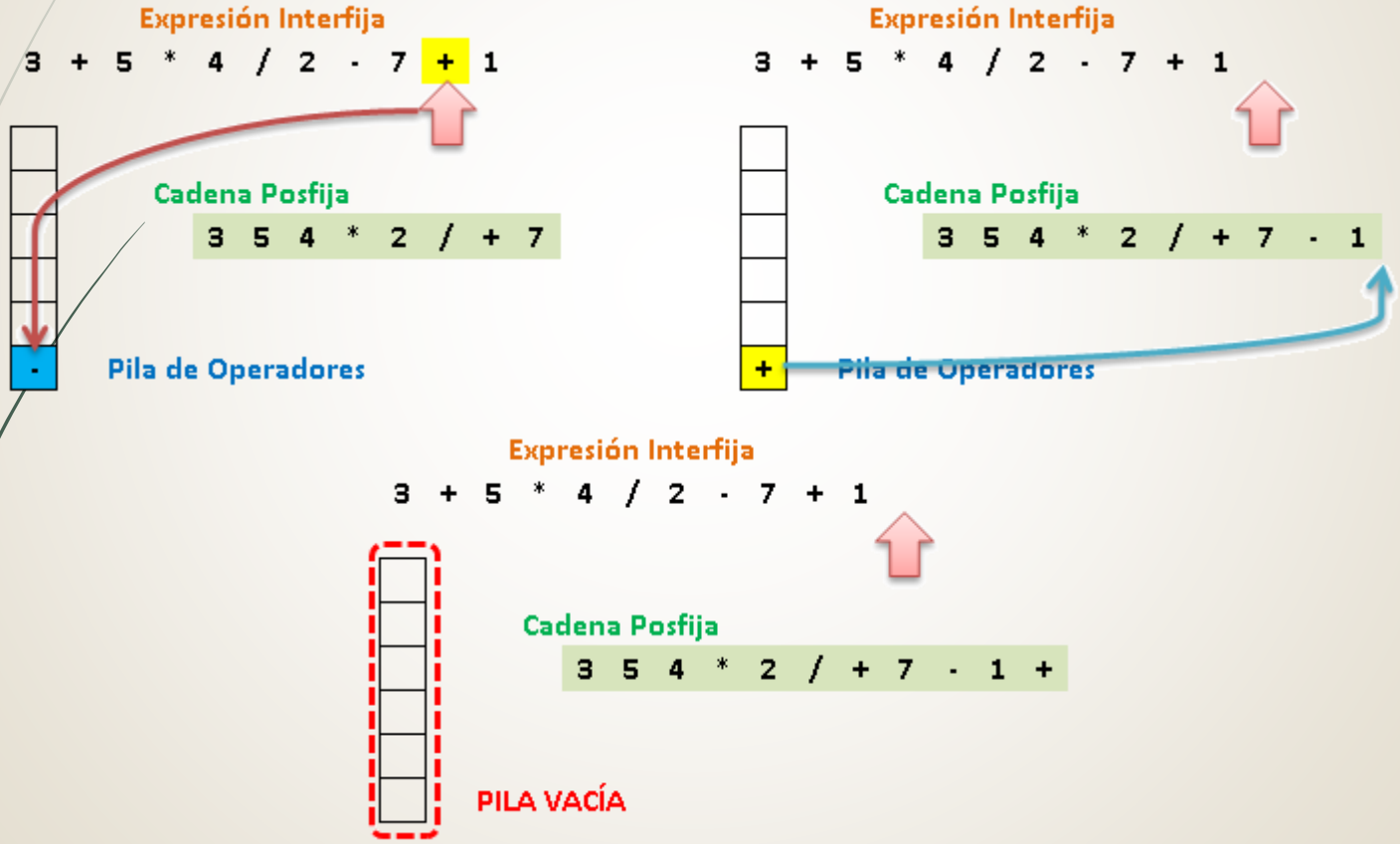
Ejemplo de Aplicación (12)

- Conversión de expresiones interfijas a posfijas



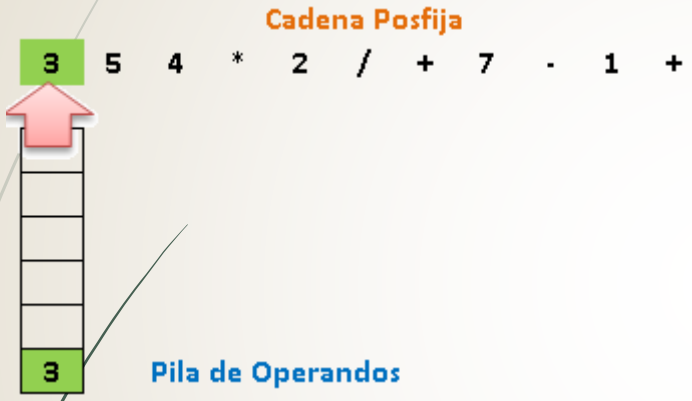
Ejemplo de Aplicación (13)

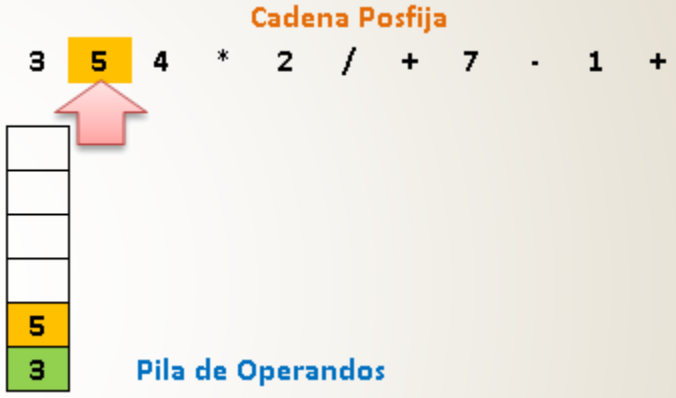
- Conversión de expresiones interfijas a posfijas

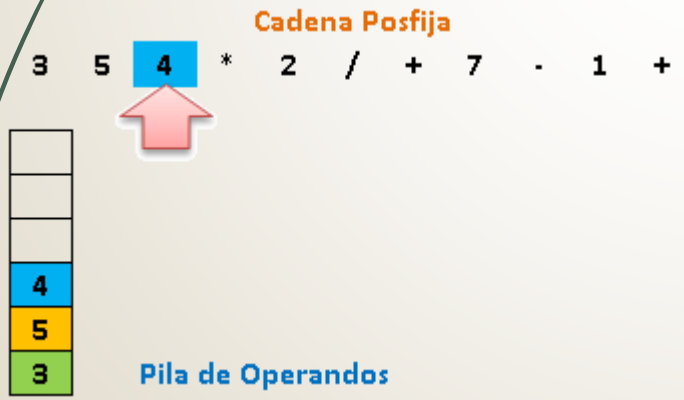


Ejemplo de Aplicación (14)

- Cálculo de expresiones posfijas

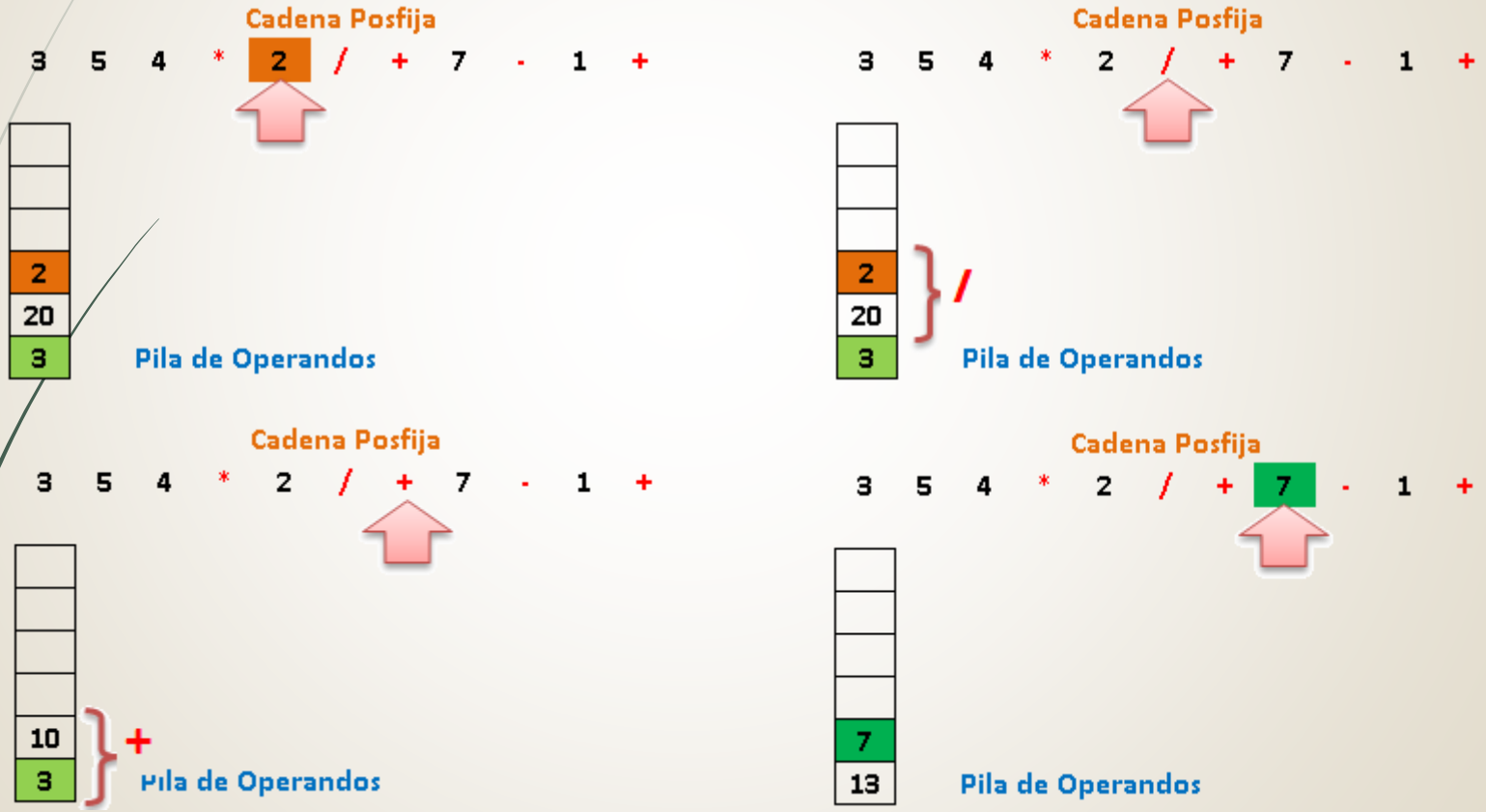






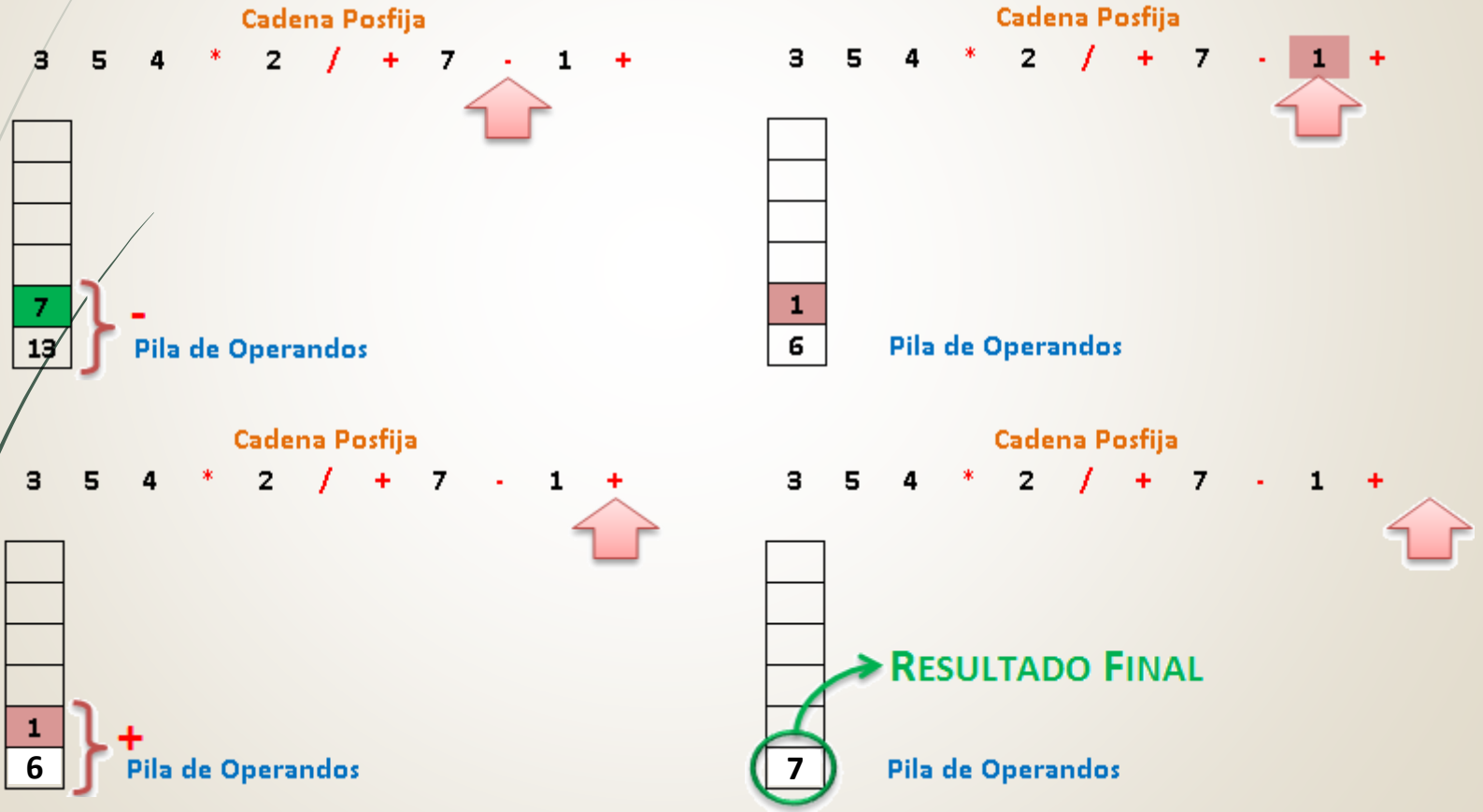
Ejemplo de Aplicación (15)

- 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.