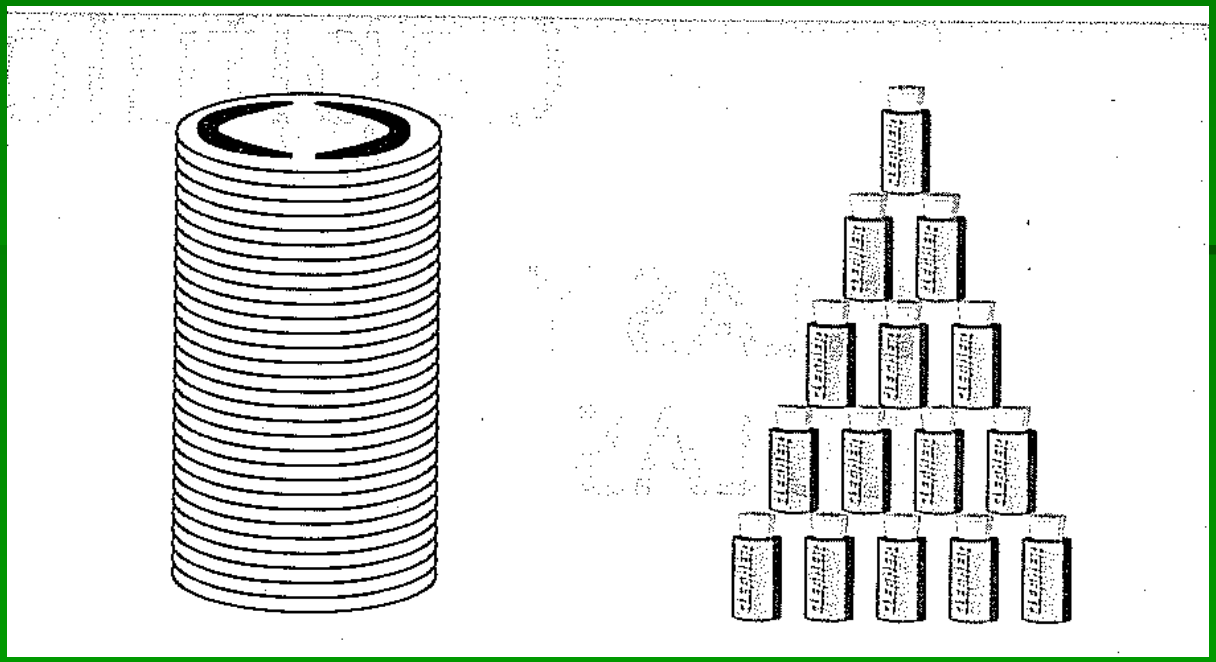
Pilas y Colas

Capítulo 3

Pilas

- Una pila representa una estructura lineal de datos en que se puede agregar o quitar elementos únicamente por uno de los dos extremos. En consecuencia, los elementos de una pila se eliminan en el orden inverso al que se insertaron. Debido a está característica, se le conoce como estructura LIFO (last input, first output).

- Existen muchos casos prácticos en los que se utiliza la idea de pila:
- Ejemplo; pila de platos, en el supermercado latas.
- Las pilas con estructuras lineales como los arreglos, ya que sus componentes ocupan lugares sucesivos en la ED y c/u tienen un único sucesor/predecesor, con excepción del primero/último.

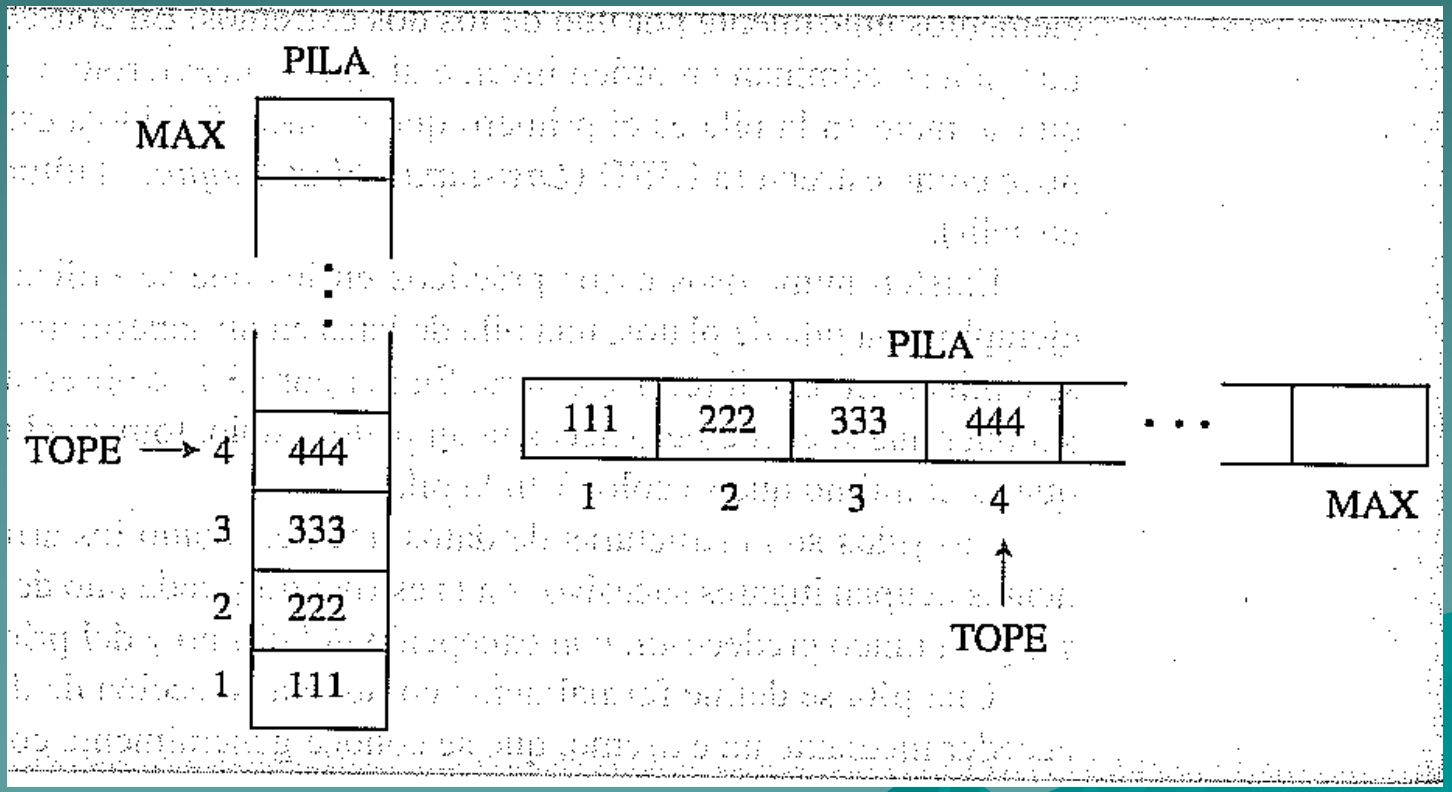


- Definición de Pila
- Una colección de datos a los cuales se les puede acceder mediante un extremo, que se conoce generalmente como tope.

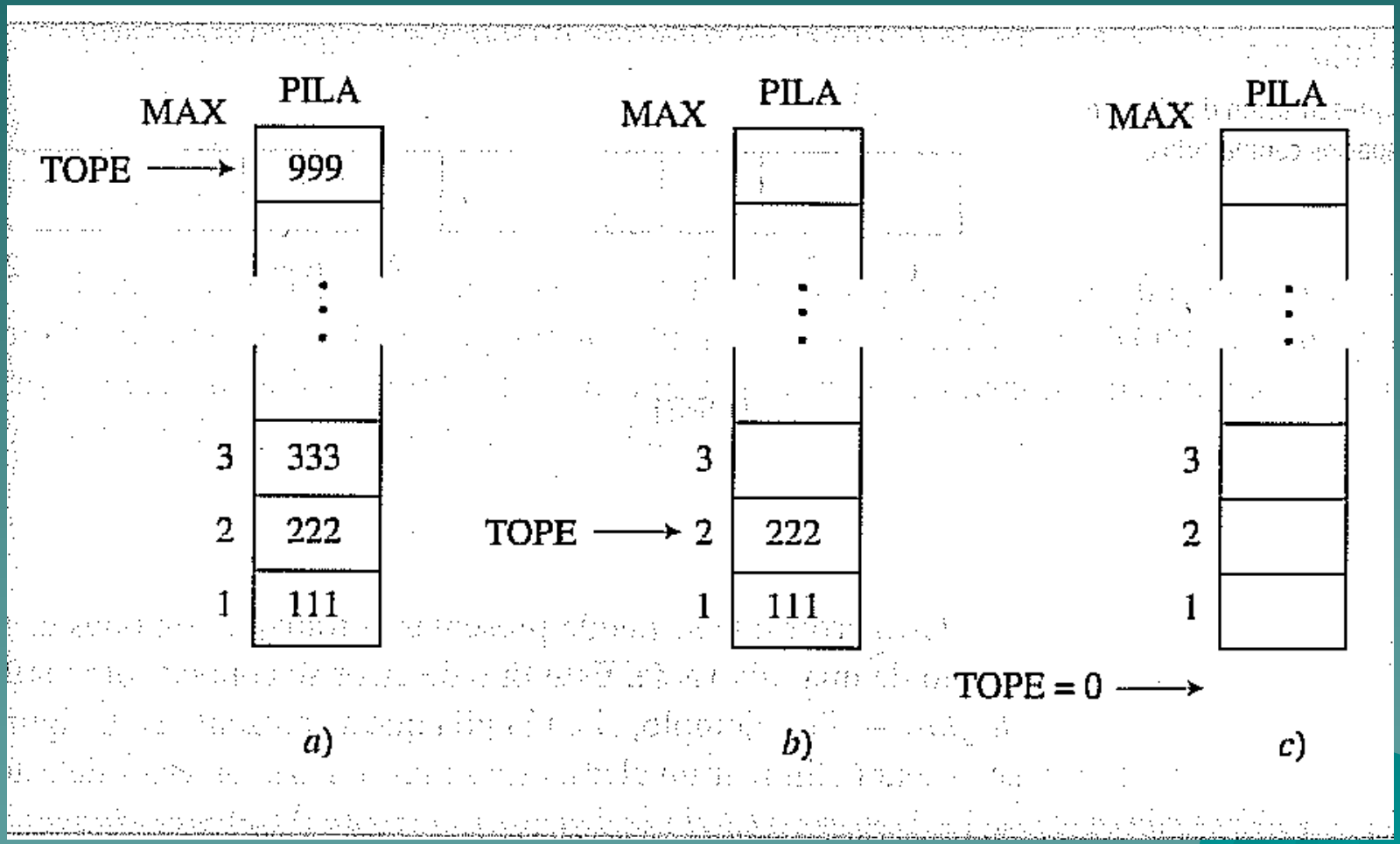
Representación de Pilas

- Las pilas no son estructuras fundamentales de datos; es decir no están definidas como tales en los lenguajes de programación. Para su representación requieren de otras EDs, como:
- Arreglos
- Listas

- OC/SG utilizan arreglos. Es importante definir el tamaño de la máximo de la pila, así como una variable auxiliar que se denomina TOPE. Está variable se utiliza para indicar el último elemento que se insertó en la pila. (Figs.)



PILAS


MAX
PILA
TOPE
999
3
333
2
222
1
111
a)
MAX
PILA
3
TOPE
2
222
1
111
b)
MAX
PILA
3
2
1
TOPE = 0
c)

Representación de pilas

- Al utilizar arreglos para implementar pilas se tiene la limitación de que se debe reservar el espacio en memoria con anticipación. Una vez dado un máximo de capacidad a la pila no es posible insertar un número de elementos mayor que el máximo establecido. Si esto ocurre, en otras palabras si la pila esta llena y se intenta insertar un nuevo elemento, se producirá un error conocido como **desbordamiento –overflow**

Posibles soluciones

- Una posible solución a este tipo de inconvenientes consiste en definir pilas de gran tamaño, pero esto resultará ineficiente y costoso si solo se utilizarán algunos elementos. No siempre es viable saber con exactitud el número de elementos a tratar, y siempre existe la posibilidad de que ocurra un error de desbordamiento.

Otra solución

- Consiste en usar **espacios compartidos** de memoria para la implementación de pilas. Supongamos que se necesitan dos pilas, c/u con un tamaño máximo de N elementos. Se definirá entonces un arreglo unidimensional de 2*N elementos, en lugar de 2 arreglos de N elementos c/u.

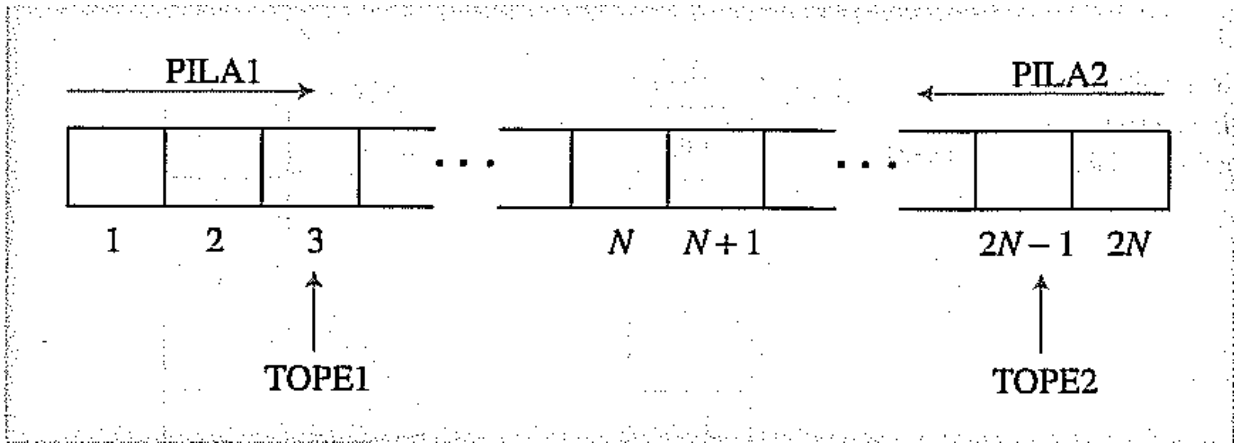
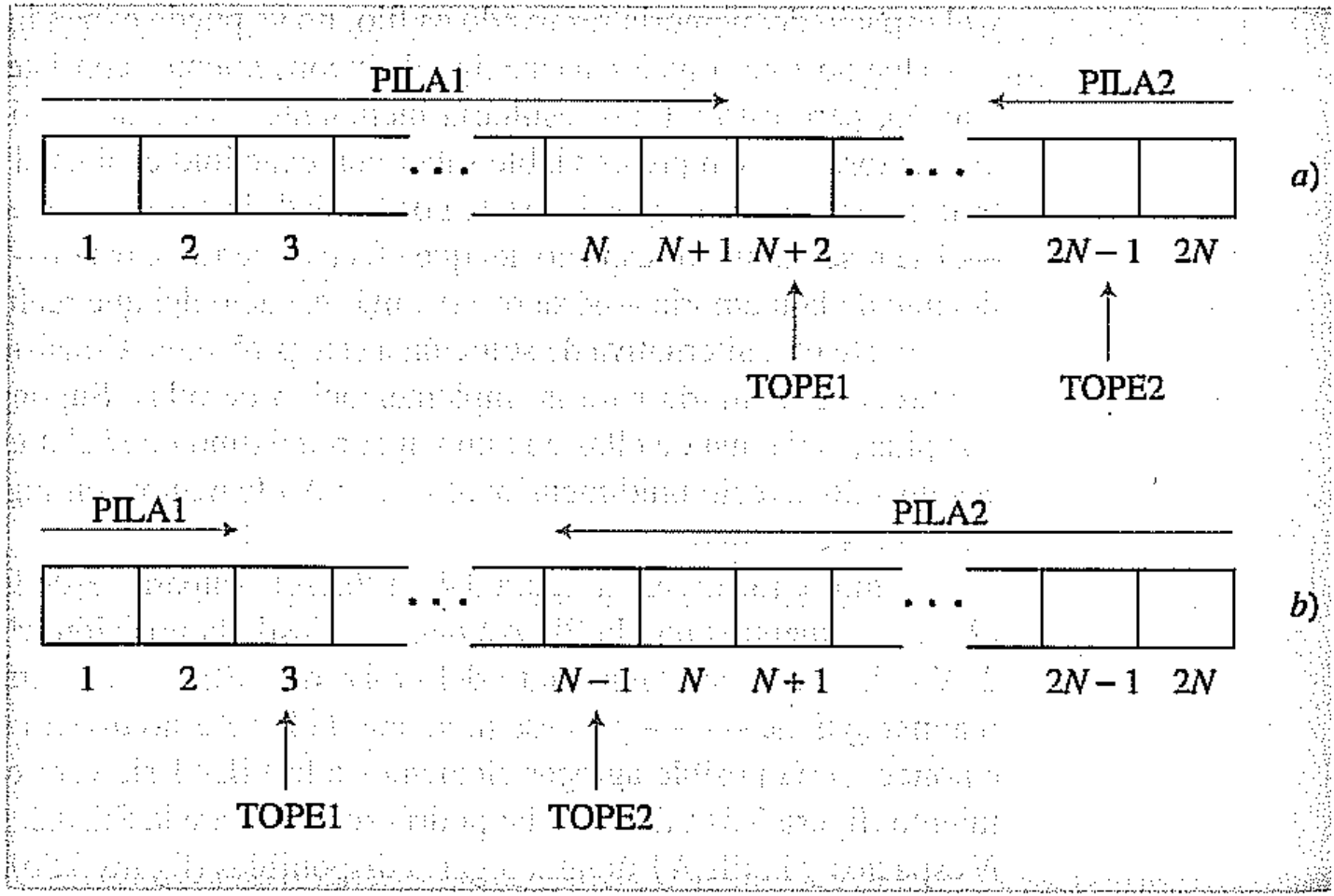


Figura muestra la PILA 1 ocupará desde la posición 1 en adelante, mientras que la PILA 2 ocupará desde la posición 2*N hacias atrás (2*N-1..)
Si en algún momento del proceso la PILA 1 necesitará más espacio del que realmente tiene -N- y en ese momento la PILA 2 no tiene ocupados todos sus N lugares entonces sería posible agregar elementos a la PILA1 sin caer en un error de desbordamiento. Algo similar podría suceder con la PILA 2.

Espacios Compartidos

FIGURA 3.5

Representación de pilas en espacios compartidos. *a*) PILA1 tiene más de *N* elementos y PILA2 tiene menos de *N* elementos. *b*) PILA2 tiene más de *N* elementos y PILA1 tiene menos de *N* elementos.



Error y Operaciones

- Otro error que se puede presentar es tratar de eliminar un elemento de un pila vacía. Este tipo de error se le conoce como **subdesbordamiento** –underflow-
- **Operaciones con pilas**
- Insertar un elemento –push- en la pila
- Eliminar -pop – de la pila
- Pila_vacía
- Pila_llena

- Considerando que se tiene una pila con una capacidad para almacenar un núm máximo de elementos –MAX- y que el último de ellos se indica con TOPE, a continuación se presentan los algoritmos de las operaciones mencionadas. Si la pila está vacía, entonces TOPE es igual a 0.

Algoritmo Pila_vacía

Pila_vacía (PILA, TOPE, BAND)

{Este algoritmo verifica si una estructura tipo pila —PILA— está vacía, asignando a BAND el valor de verdad correspondiente. La pila se implementa en un arreglo unidimensional. TOPE es un parámetro de tipo entero. BAND es un parámetro de tipo booleano}

1. *Si* (TOPE = 0) {Verifica si no hay elementos almacenados en la pila}
   entonces
      Hacer BAND ← VERDADERO {La pila está vacía}
   si no
      Hacer BAND ← FALSO {La pila no está vacía}
2. {Fin del condicional del paso 1}

Algoritmo Pila_llena

Pila_llena (PILA, TOPE, MAX, BAND)

{Este algoritmo verifica si una estructura tipo pila —PILA— está llena, asignando a BAND el valor de verdad correspondiente. La pila se implementa en un arreglo unidimensional de MAX elementos. TOPE es un parámetro de tipo entero. BAND es un parámetro de tipo booleano}

1. *Si* (TOPE = MAX)
   entonces
      Hacer BAND ← VERDADERO {La pila está llena}
   si no
      Hacer BAND ← FALSO {La pila no está llena}
2. {Fin del condicional del paso 1}

Algoritmo Pone

Pone (PILA, TOPE, MAX, DATO)

{Este algoritmo agrega el elemento DATO en una estructura tipo pila —PILA—, si la misma no está llena. Actualiza el valor de TOPE. MAX representa el número máximo de elementos que puede almacenar PILA. TOPE es un parámetro de tipo entero}

1. Llamar a Pila_llena con PILA, TOPE, MAX y BAND
2. *Si* (BAND = VERDADERO)
   entonces
      Escribir "Desbordamiento – Pila llena"
   si no
      Hacer TOPE ← TOPE + 1 y PILA[TOPE] ← DATO
      {Actualiza TOPE e inserta el nuevo elemento en el TOPE de PILA}
3. {Fin del condicional del paso 2}

Algoritmo Quita

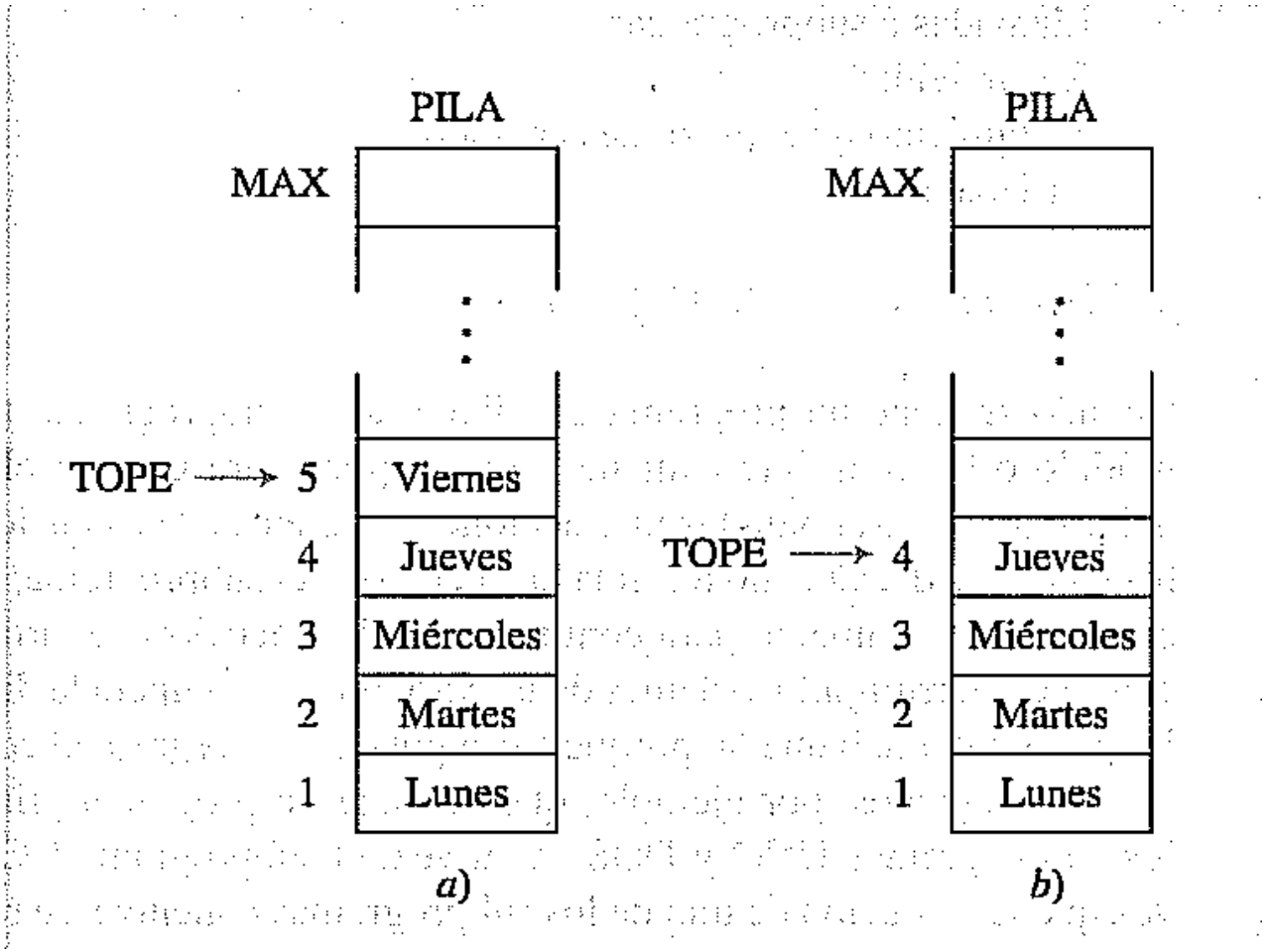
Quita (PILA, TOPE, DATO)

{Este algoritmo saca un elemento —DATO— de una estructura tipo pila —PILA—, si ésta no se encuentra vacía. El elemento que se elimina es el que se encuentra en la posición indicada por TOPE}

Ejemplo Días de la semana

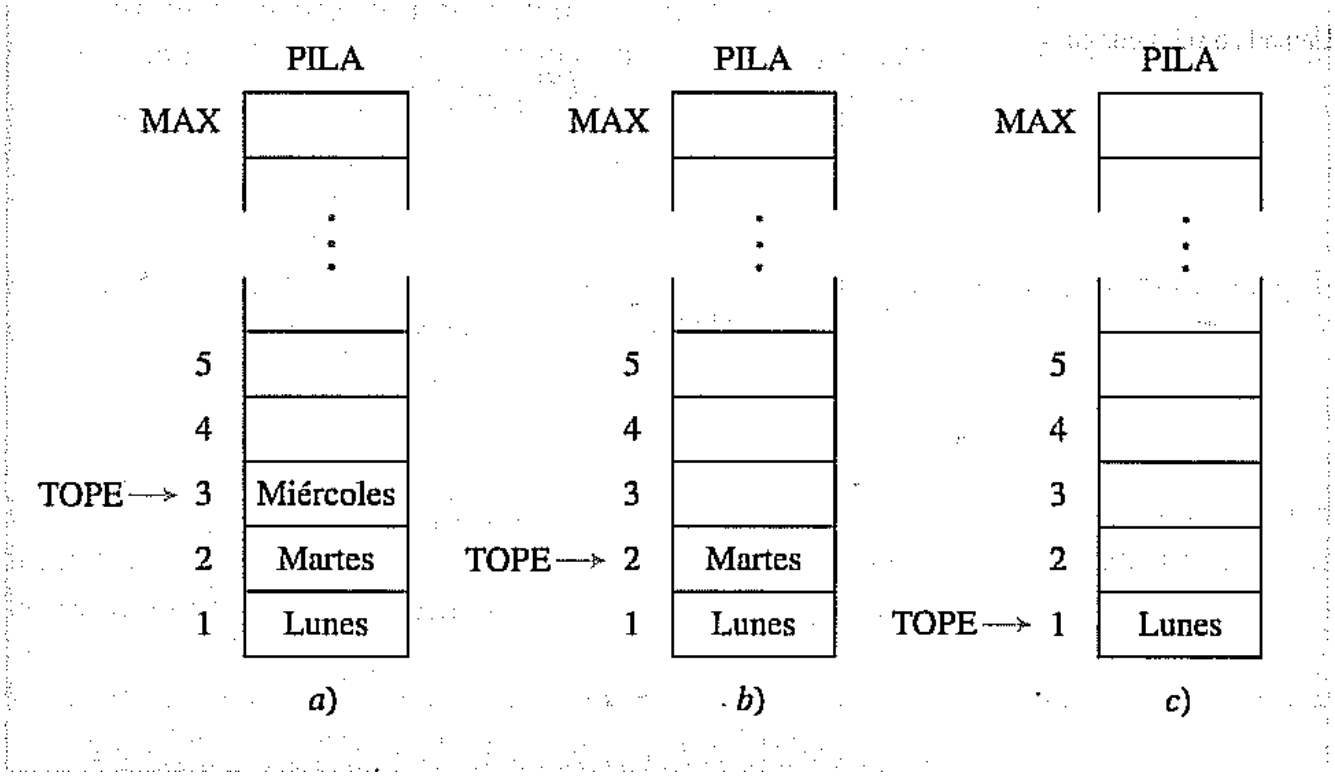
1. Llamar a Pila_vacía con PILA, TOPE y BAND
2. *Si* (BAND = VERDADERO)
   entonces
      Escribir "Subdesbordamiento – Pila vacía"
   si no
      Hacer DATO ← PILA [TOPE] y TOPE ← TOPE – 1 {Actualiza TOPE}
3. {Fin del condicional del paso 2}

Ejemplo Días de la semana



Ejemplo Días de la semana

FIGURA 3.7
Inserción y eliminación, *a*) Luego de sacar jueves. *b*) Luego de sacar miércoles. *c*) Luego de sacar martes.



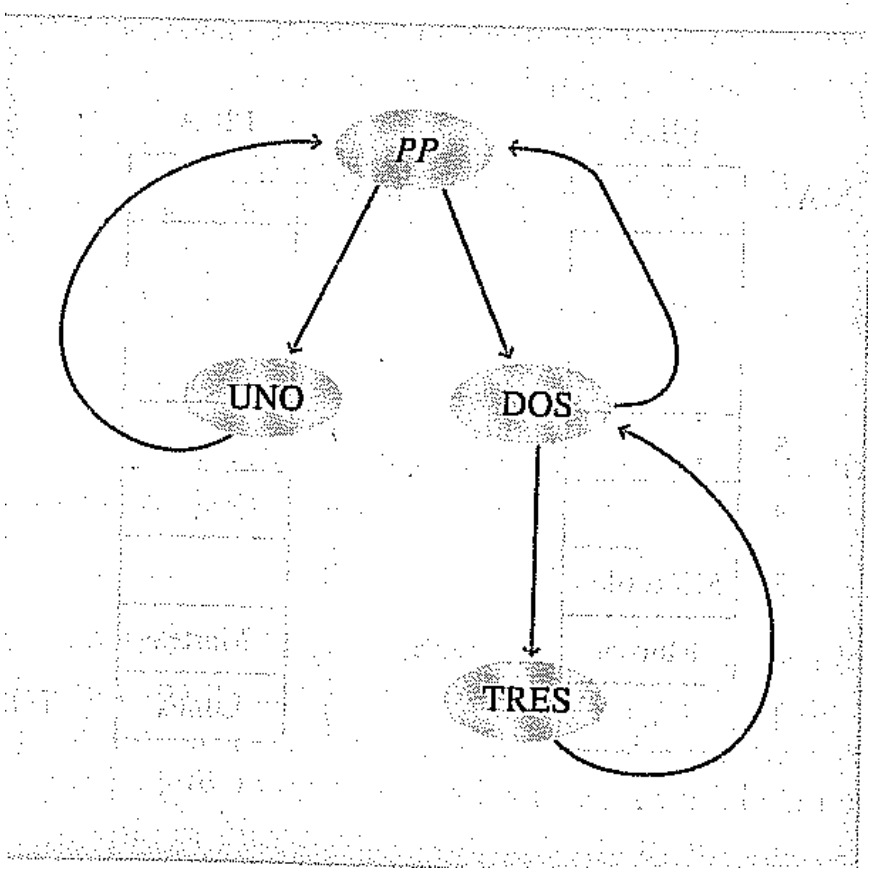
Aplicaciones de Pilas

- Las pilas son un EDs muy usadas en la solución de diversos tipos de problemas, en el área de computación. Algunos de los casos más representativos de aplicación de las mismas son:
- Llamadas a subprogramas
- Recursividad
- Tratamiento de expresiones aritméticas
- Ordenación

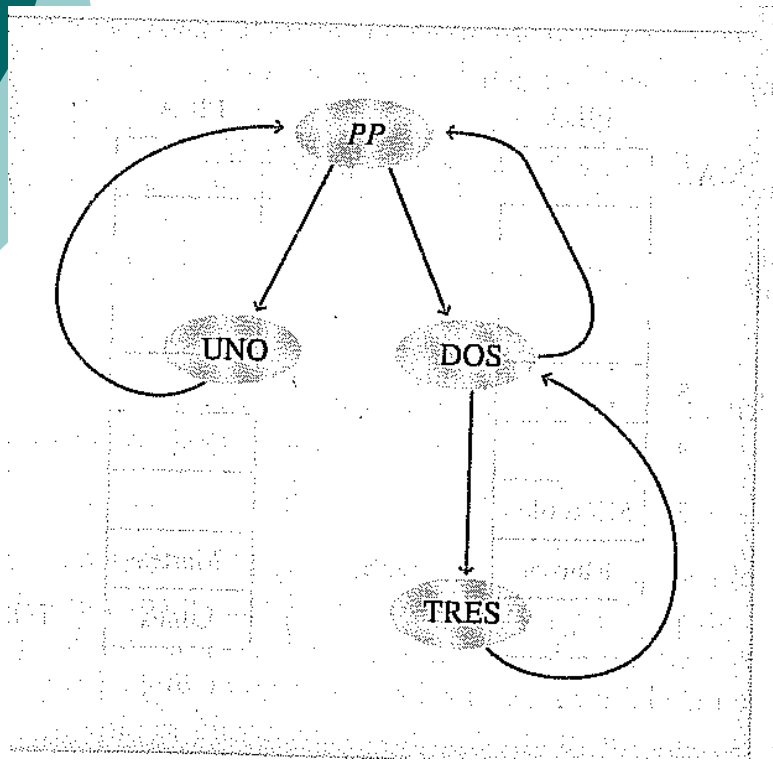
Llamadas a subprogramas

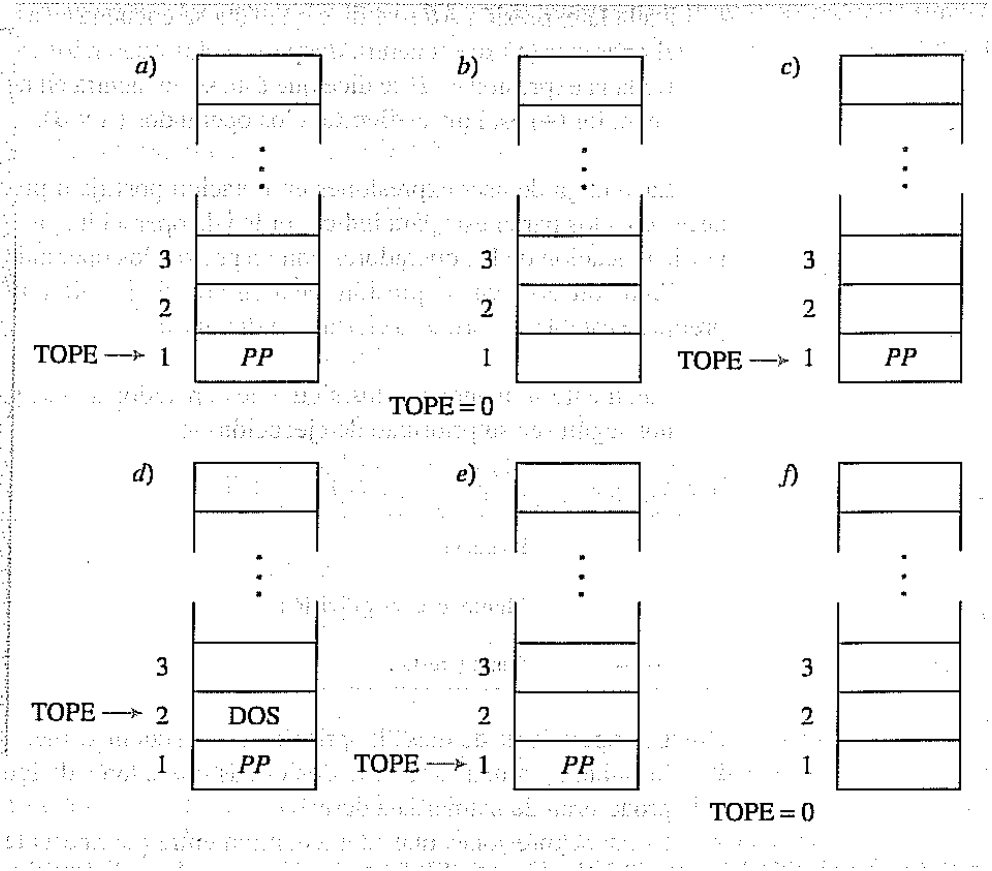
- Cuando se tiene un programa que llama a un subprograma, también conocido como módulo o función, internamente se usan pilas para guardar el estado de las variables del programa, así como instrucciones pendientes de ejecución en el momento que se hace la llamada.
- Cuando termina la ejecución del subprograma,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

Ejemplo:



a) Se guarda la posición en la que se hizo la llamada. Al terminar UNO, el control se regresa a PP recuperando previamente la dirección de la pila b)





Conclusiones

- Finalmente podemos concluir que las pilas son necesarias en este tipo de aplicaciones por lo siguiente;
- Permiten guardar la dirección del programa, o subprograma, desde donde se hizo la llamada a otros subprogramas, para regresar posteriormente y seguir ejecutándolo a partir de la instrucción inmediata a la llamada.
- Permiten guardar el estado de las variables en el momento en que se hace la llamada, para seguir ocupándolas al regresar del subprograma.

Tratamiento de expresiones aritméticas

- Un problema interesante en computación consiste en convertir expresiones en notación infija a su equivalente en notación prefija o posfija.

- Dada la expresión **A+B** se dice que ésta en notación **infija**, porque el operador **(+)** se encuentra **entre** los operandos (A y B).

- Dada la expresión **AB+** se dice que ésta en notación **posfija** por que..

- Dada la expresión +AB se dice que ésta en notación **prefija.**.

- La ventaja de usar expresiones en notación posfija o prefija radica en que no es necesario utilizar paréntesis para indicar orden de operación, ya que éste queda establecido por la ubicación de los operadores con respecto a los operandos.

Tratamiento de expresiones aritméticas

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

Operador	Nombre de la operación
^	Potencia
* /	Multiplicación y división
+ −	Suma y resta

Ejemplos

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

TABLA 3.1

Traducción de infija a posfija

Paso	Expresión
0	$X + Z * W$
1	$X + ZW*$,
2	$XZW* +$

EJEMPLO 2

b) Expresión infija: $(X + Z)^* \; W / T \; \hat{} \; Y - V$

Paso	Expresión
0	$(X + Z)^* \; W/T \; \hat{} \; Y - V$

Algoritmo Convierta a notación posfija

Conv_postfija (EI, EPOS)

{Este algoritmo traduce una expresión infija —*EI*— a postfija —EPOS—, haciendo uso de una pila —PILA—. MAX es el número máximo de elementos que puede almacenar la pila}

1. Hacer TOPE ← 0
2. Mientras (*EI* sea diferente de la cadena vacía) Repetir
   Tomar el símbolo más a la izquierda de *EI*. Recortar luego la expresión
   2.1 *Si* (el símbolo es paréntesis izquierdo)
   entonces {Poner símbolo en PILA. Se asume que hay espacio en PILA}
   Llamar a Pone con PILA, TOPE, MAX y símbolo
   si no
   2.1.1 *Si* (el símbolo es paréntesis derecho)
   entonces
   2.1.1.1 Mientras (PILA[TOPE] ≠ paréntesis izquierdo) Repetir
   Llamar a Quita con PILA, TOPE y DATO
   Hacer EPOS ← EPOS + DATO

Continuación del Algoritmo Convierta a notación posfija

2.1.1.2 {Fin del ciclo del paso 2.1.1.1}
Llamar a Quita con PILA, TOPE y DATO
{Se quita el paréntesis izquierdo de PILA y no se agrega a EPOS}
si no
2.1.1.3 *Si* (el símbolo es un operando)
entonces
Agregar símbolo a EPOS
si no {Es un operador}
Llamar Pila_vacía con PILA, TOPE y BAND
2.1.1.3A Mientras (BAND = FALSO) y (la prioridad del operador sea menor o igual que la prioridad del operador que está en la cima de PILA)
Repetir
Llamar a Quita con PILA, TOPE y DATO
Hacer EPOS ← EPOS + DATO
Llamar a Pila_vacía con PILA, TOPE y BAND
2.1.1.3B {Fin del ciclo del paso 2.1.1.3A}
Llamar a Pone con PILA, TOPE, MAX y símbolo
2.1.1.4 {Fin del condicional del paso 2.1.1.3}
2.1.2 {Fin del condicional del paso 2.1.1}
2.2 {Fin del condicional del paso 2.1}
3. {Fin del ciclo del paso 2}
4. Llamar a Pila_vacía con PILA, TOPE y BAND
5. Mientras (BAND = FALSO) Repetir
Llamar a Quita con PILA, TOPE y DATO
Hacer EPOS ← EPOS + DATO
Llamar a Pila_vacía con PILA, TOPE y BAND
6. {Fin del ciclo del paso 5}
7. Escribir EPOS

Ejemplo 3 Para ilustrar el funcionamiento del Algoritmo Conv_posfija

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

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 * +

Notación Infija

- Ejemplo1
- Expresión infija: X + Z * W
- Expresión prefija +X*ZW

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

Ejemplo 2

Expresión infija: $(X + Z)^* W / T \wedge Y - V$

Paso	Expresión
0	$(X + Z) * W / T \wedge Y - V$

Algoritmo Convertir a prefija

Conv_prefija (EI, EPRE)

{Este algoritmo traduce una expresión en notación infija, EI a prefija, EPRE, haciendo uso de una pila —PILA—}

{TOPE es una variable de tipo entero y MAX representa el máximo número de elementos que puede almacenar la pila}

1. Hacer TOPE ← 0
2. Mientras (*EI* sea diferente de la cadena vacía) *Repetir*
   Tomar el símbolo más a la derecha de *EI* recortando luego la expresión
   2.1 *Si* (el símbolo es paréntesis derecho)
   entonces {Poner símbolo en pila}
   Llamar a Pone con PILA, TOPE, MAX y símbolo
   si no
   2.1.1 Si (símbolo es paréntesis izquierdo)
   entonces
   2.1.1.1 Mientras (PILA[TOPE] ≠ paréntesis derecho) *Repetir*
   Llamar a Quita con PILA, TOPE y DATO
   Hacer EPRE ← EPRE + DATO
   2.1.1.2 {Fin del ciclo del paso 2.1.1.1}
   {Sacamos el paréntesis derecho de PILA y no se agrega a EPRE}
   Llamar a Quita con PILA, TOPE y DATO
   si no
   2.1.1.3 *Si* (símbolo es un operando)
   entonces
   Agregar símbolo a EPRE
   si no {Es un operador}
   Llamar a Pila_vacía con PILA, TOPE y BAND
   2.1.1.3A Mientras (BAND = FALSO) y (la prioridad del operador sea menor que la prioridad del operador que está en la cima de PILA) *Repetir*
   Llamar a Quita con PILA, TOPE y DATO
   Hacer EPRE ← EPRE + DATO
   Llamar a Pila_vacía con PILA, TOPE y BAND
   2.1.1.3B {Fin del ciclo del paso 2.1.1.3A}
   Llamar a Pone con PILA, TOPE, MAX y símbolo
   2.1.1.4 {Fin del condicional del paso 2.1.1.3}
   2.1.2 {Fin del condicional del paso 2.1.1}
   2.2 {Fin del condicional del paso 2.1}
3. {Fin del ciclo del paso 2}
   Llamar a Pila_vacía con PILA, TOPE y BAND
4. Mientras (BAND = FALSO) *Repetir*
   Llamar a Quita con PILA, TOPE y DATO
   Hacer EPRE ← EPRE + DATO
   Llamar a Pila_vacía con PILA, TOPE y BAND
5. {Fin del ciclo del paso 4}
6. Escribir EPRE en forma invertida

Funcionamiento del Algoritmo

Ejemplo3

Expresión infija: X + Z * W

Expresión prefija +X*ZW

Paso	EI	Símbolo analizado	Pila	EPRE
0	X + Z * W			
1	X + Z *	W		W
2	X + Z	*	*	W
3	X +	Z	*	WZ
4	X	+		WZ *
		+	+	WZ *
5		X	+	WZ * X
6				WZ * X +
7		Invertir EPRE:		+ X * ZW

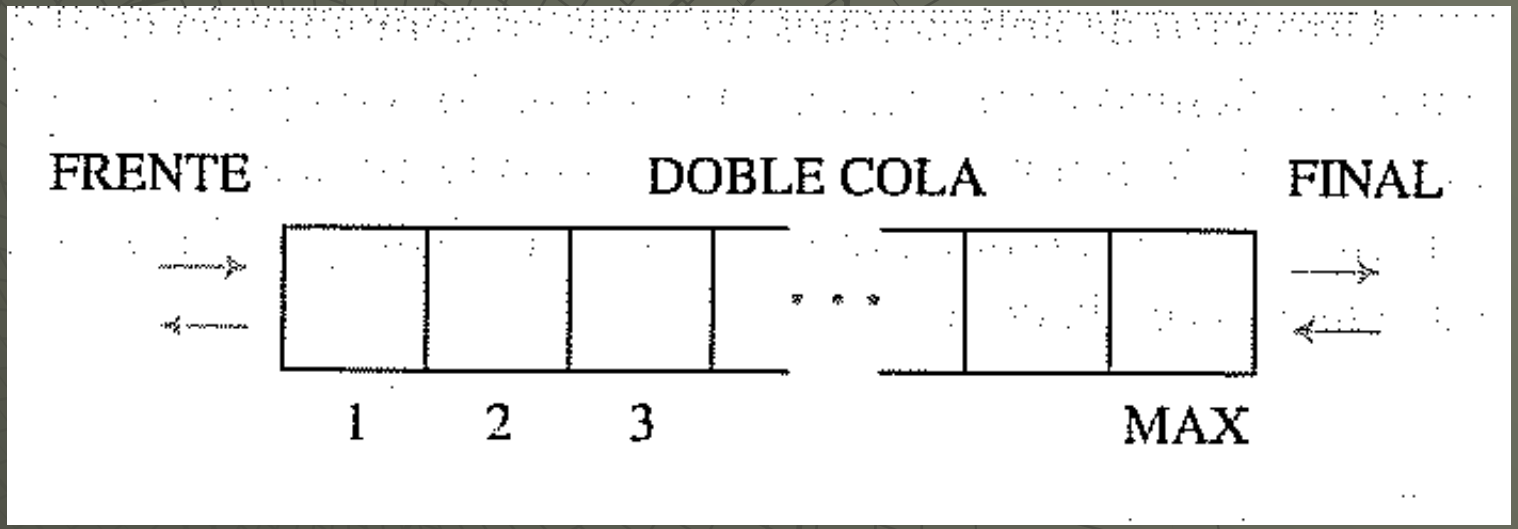
Ejemplo 4

Expresión infija: $(X + Z)^* W / T \wedge Y - V$

Paso	Izq	Símbolo analizado	Pila	Derecha
0	$(X + Z) * W / T \wedge Y - V$			

Doble Cola

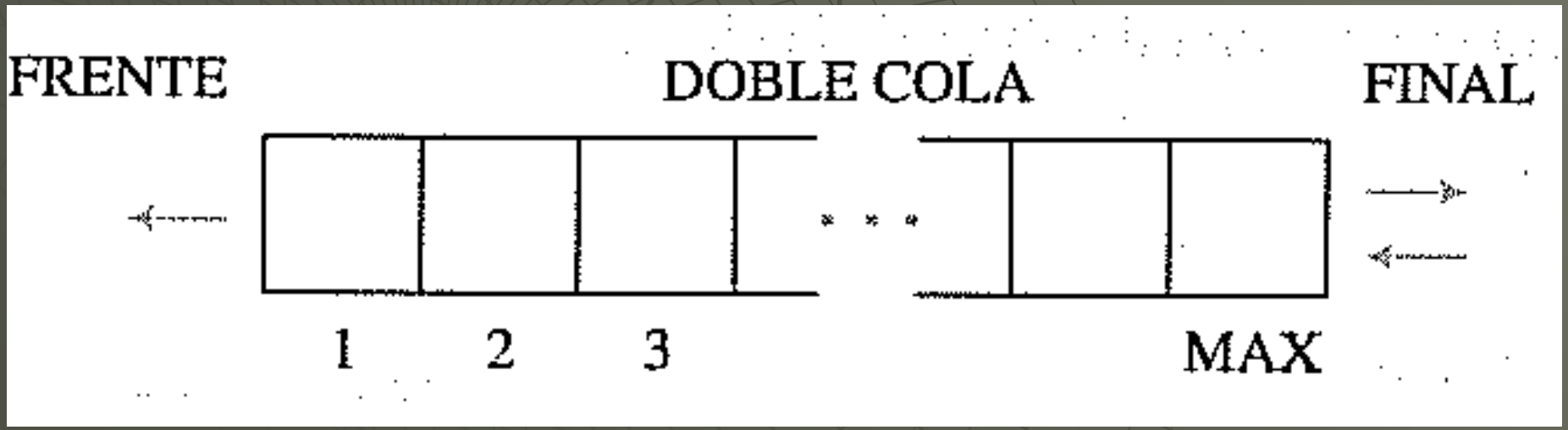
- Una doble cola o bicola es una generalización de una ED tipo cola. En una doble cola, los elementos se pueden insertar o eliminar por cualquiera de los dos extremos. Es decir, se pueden insertar y eliminar valores tanto por FRENTE como por el FINAL de la cola.



Existen dos variantes de las dobles colas:

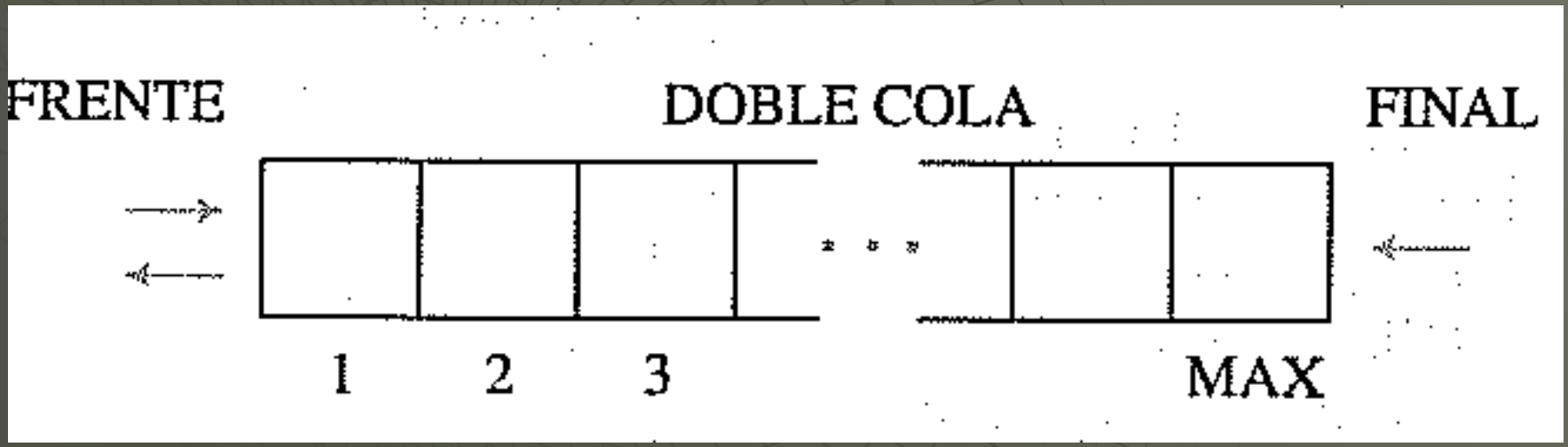
- Doble cola (DC) con entrada restringida

La primera permite hacer eliminaciones
por cualquiera de los dos extremos,
mientras que las inserciones sólo por el FINAL de la cola



DC con salida restringida

- La segunda variante permite que las inserciones se realicen por cualquiera de los 2 extremos, mientras que las eliminaciones sólo por FRENTE de la cola.

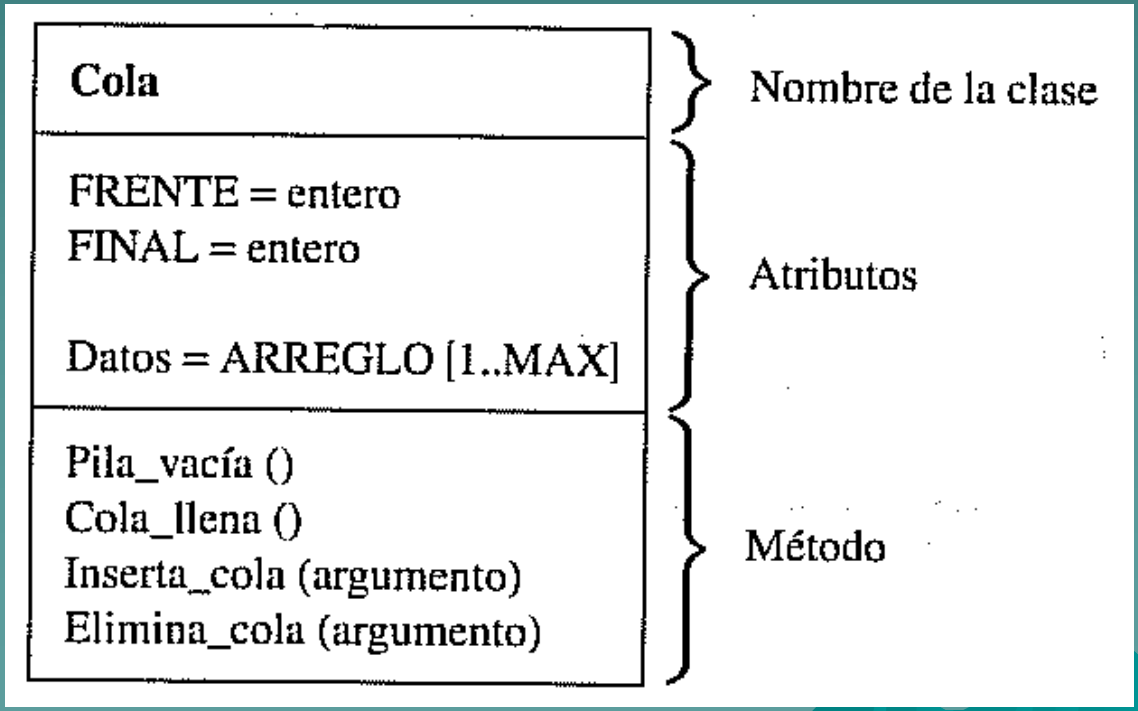


Aplicaciones de las colas

- El concepto de cola está ligado a computación.
- Impresión.
- Otro en sistemas de tiempo compartido (memoria)

La clase Cola

- La clase cola tiene atributos y métodos, como cualquier clase.



- Se tiene acceso a los miembros de un objeto de la clase cola por medio de la notación de puntos. (COOBJ)
- COOBJ.Cola_llena
- COOBJ.Cola_inserta (argumento)

Itinerario del 4 de Septiembre 2008

- Revisar programas de tarea (2 semanas)
- Programas de notación posfija y prefija