Capítulo 3

PILAS Y COLAS

3.1 INTRODUCCIÓN

Cuando se presentaron los arreglos, en el capítulo 1, se mencionó que eran estructuras lineales. Es decir, cada componente tiene un único sucesor y un único predecesor con excepción del primero y del último, respectivamente. Por otra parte, al analizar las operaciones de inserción y eliminación, se observó que los elementos se podían insertar o eliminar en cualquier posición del arreglo. Cabe señalar, sin embargo, que existen problemas que por su naturaleza requieren que los elementos se agreguen o se quiten sólo por un extremo. Este capítulo se dedica al estudio de **pilas** y **colas**, que son estructuras de datos lineales con restricciones en cuanto a la posición en la cual se pueden llevar a cabo las operaciones de inserción y eliminación de componentes.

3.2 PILAS

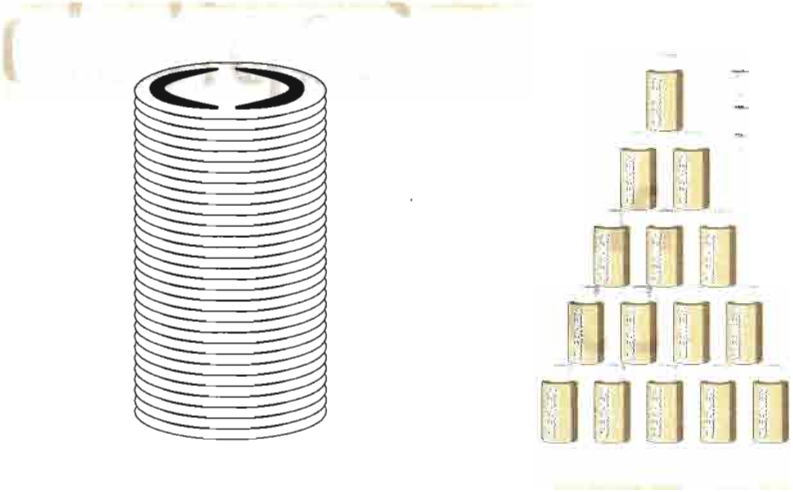
Una **pila** representa una estructura lineal de datos en la que se puede agregar o quitar elementos únicamente por uno de los dos extremos. En consecuencia, los elementos de una pila se eliminan en orden inverso al que se insertaron; es decir, el último elemento que se mete en la pila es el primero que se saca. Debido a esta característica, se le conoce como estructura **LIFO** (*Last-Input, First-Output:* el último en entrar es el primero en salir).

Existen numerosos casos prácticos en los que se utiliza el concepto de pila; por ejemplo, una pila de platos, una pila de latas en un supermercado, una pila de libros que se exhiben en una librería, etcétera. En la figura 3.1 se observa una pila de platos. Es de suponer que si el cocinero necesita un plato limpio, tomará el que está encima de todos, que es el último que se colocó en la pila.

Las **pilas** son estructuras de datos lineales, como los arreglos, ya que los componentes ocupan lugares sucesivos en la estructura y cada uno de ellos tiene un único sucesor y un único predecesor, con excepción del último y del primero, respectivamente.

Una **pila** se define formalmente como una colección de datos a los cuales se puede acceder mediante un extremo, que se conoce generalmente como tope.

FIGURA 3.1
Ejemplos prácticos de pilas.



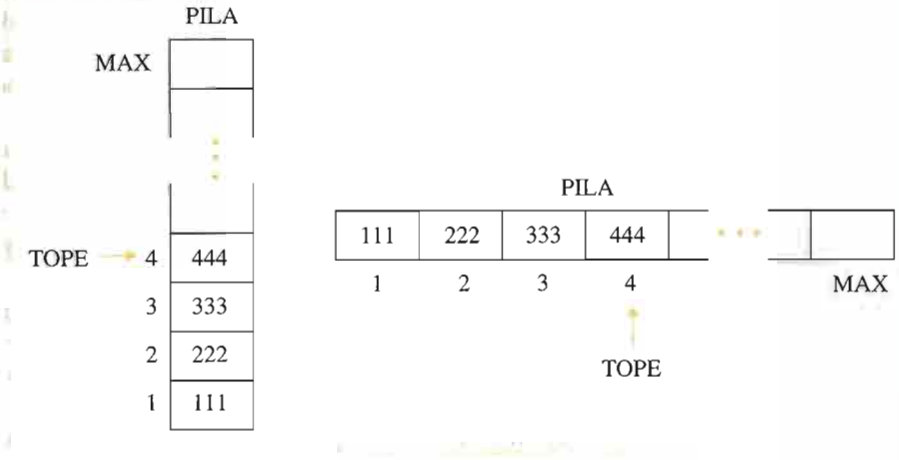
3.2.1 Representación de pilas

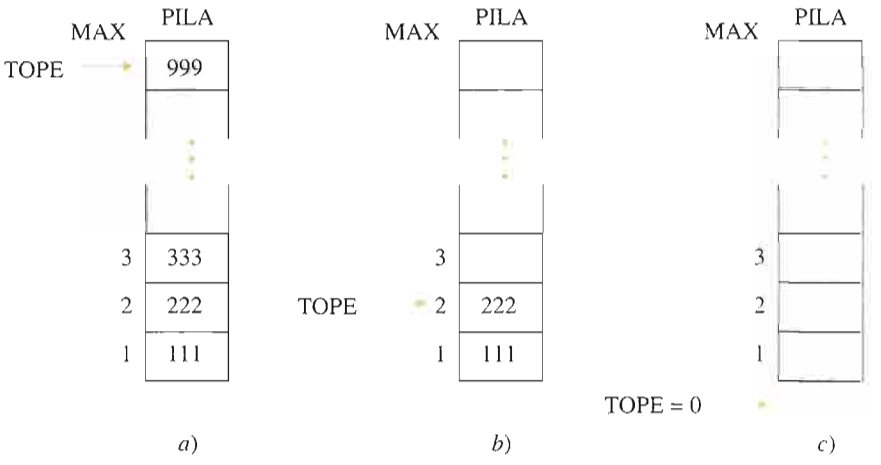
Las pilas no son estructuras fundamentales de datos; es decir, no están definidas como tales en los lenguajes de programación. Para su representación requieren el uso de otras estructuras de datos, como:

- Arreglos
- Listas

En este libro se utilizarán arreglos. En consecuencia, es importante definir el tamaño máximo de la pila, así como una variable auxiliar a la que se denomina TOPE. Esta variable se utiliza para indicar el último elemento que se insertó en la pila. En la figura 3.2 se presentan dos alternativas de representación de una pila, utilizando arreglos.

FIGURA 3.2
Representación de pilas.





En la figura 3.3 se presentan ejemplos de *a*) pila llena, *b*) pila con algunos elementos y *c*) pila vacía.

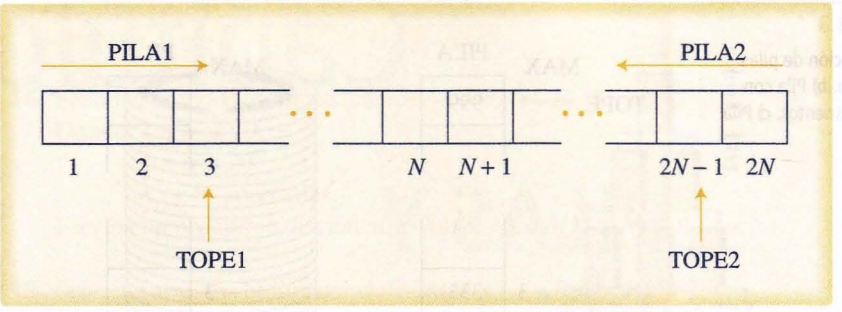
Al utilizar arreglos para implementar pilas se tiene la limitación de que se debe reservar espacio de memoria con anticipación, característica propia de los arreglos. Una vez dado un máximo de capacidad a la pila no es posible insertar un número de elementos mayor al máximo establecido. Si la pila estuviera llena y se intentara insertar un nuevo elemento, se producirá un error conocido como **desbordamiento** —*overflow*—. Por ejemplo, si en la pila que se presenta en la figura 3.3*a*, donde TOPE = MAX, se quisiera insertar un nuevo elemento, se producirá un error de este tipo. La pila está llena y el espacio de memoria reservado es fijo, no se puede expandir ni contraer.

Una posible solución a este tipo de inconvenientes consiste en definir pilas de gran tamaño, pero esto último resultaría ineficiente y costoso si sólo se utilizaran algunos elementos. No siempre es viable saber con exactitud cuál es el número de elementos a tratar; por tanto, siempre existe la posibilidad de cometer un error de desbordamiento —si se reserva menos espacio del que efectivamente se usará— o bien de hacer uso ineficiente de la memoria —si se reserva más espacio del que realmente se necesita—.

Existe otra alternativa de solución a este problema. Consiste en usar **espacios compartidos** de memoria para la implementación de pilas. Supongamos que se necesitan dos pilas, cada una de ellas con un tamaño máximo de N elementos. Se definirá entonces un solo arreglo unidimensional de 2 * N elementos, en lugar de dos arreglos de N elementos cada uno.

Como se ilustra en la figura 3.4, la PILA1 ocupará desde la posición 1 en adelante (2, 3, ...), mientras que la PILA2 ocupará desde la posición 2*N hacia atrás (2*N – 1. 2*N – 2, ...). Si en algún punto del proceso la PILA1 necesitara más espacio del que realmente tiene —N— y en ese momento la PILA2 no tuviera ocupados sus N lugares, entonces sería posible agregar elementos a la PILA1 sin caer en un error de desbordamiento (figura 3.5*a*). Algo similar podría suceder para la PILA2, si ésta necesitara más de N espacios y la PILA1 tuviera lugares disponibles (figura 3.5*b*).

FIGURA 3.4
Representación de pilas en espacios compartidos.



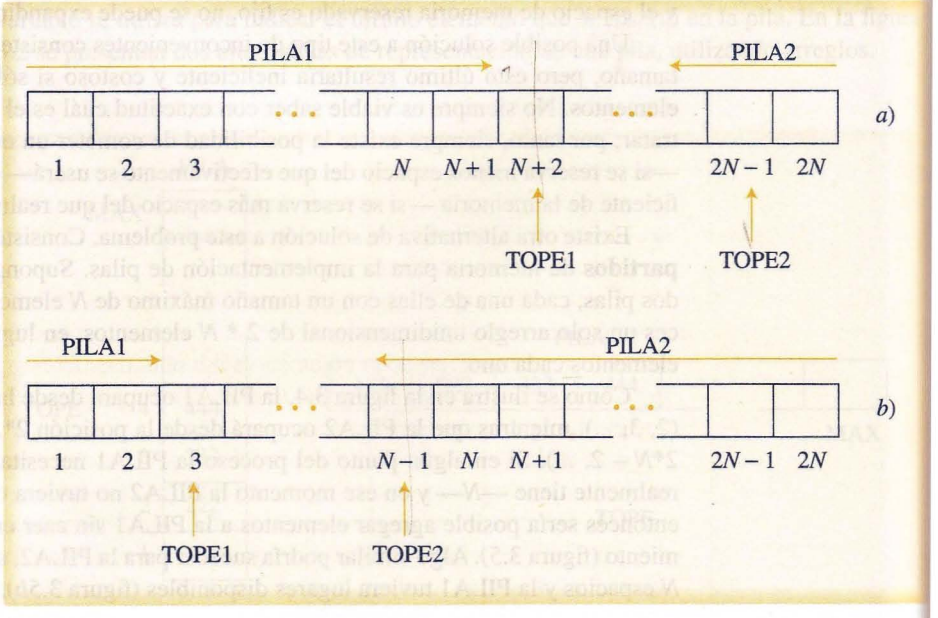
Otro error que se puede presentar al trabajar con pilas es tratar de eliminar un elemento de una pila vacía. Este tipo de error se conoce cómo **subdesbordamiento** —*underflow*—. Por ejemplo, si en la pila que se presenta en la figura 3.3*c*, donde TOPE < 1, se deseara eliminar un elemento, se presentaría un error de este tipo.

3.2.2 Operaciones con pilas

La definición de una estructura de datos queda completa al incluir las operaciones que se pueden realizar en ella. Para el caso de las pilas, las operaciones básicas que se pueden llevar a cabo son:

- Insertar un elemento —*Push*— en la pila
- Eliminar un elemento —*Pop*— de la pila

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.



Y las operaciones auxiliares:

- Pila_vacía
- Pila_llena

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

Algoritmo 3.1 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 3.2 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 3.3 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 3.4 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}

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}

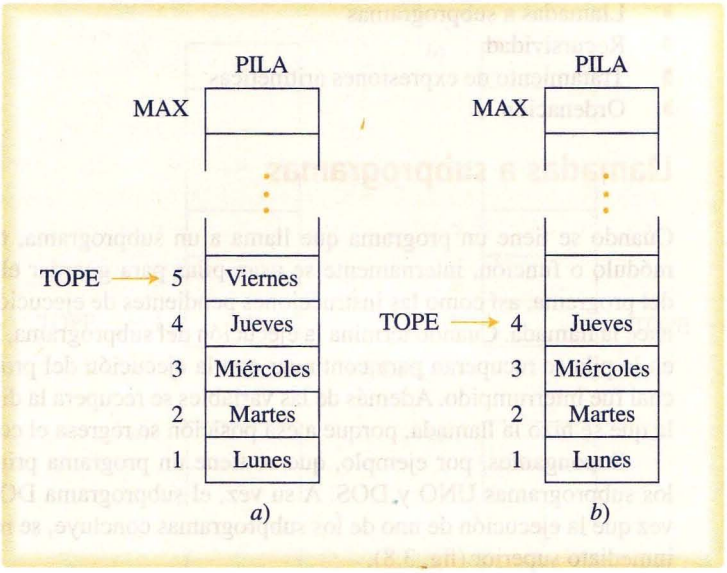
A continuación se presenta un ejemplo para ilustrar el funcionamiento de las operaciones de inserción y eliminación en pilas.

Ejemplo 3.1

Si se insertaran los elementos *lunes*, *martes*, *miércoles*, *jueves* y *viernes* en PILA, la estructura quedaría tal y como se muestra en la figura 3.6*a*. Ahora bien, si se eliminara el elemento *viernes*, el TOPE apuntaría ahora a jueves (fig. 3.6*b*).

Si en algún momento se quisiera eliminar al elemento *martes*, esto no sería posible ya que sólo se puede tener acceso al elemento que se encuentra en la cima de la pila.

FIGURA 3.6
Inserción y eliminación en pilas.

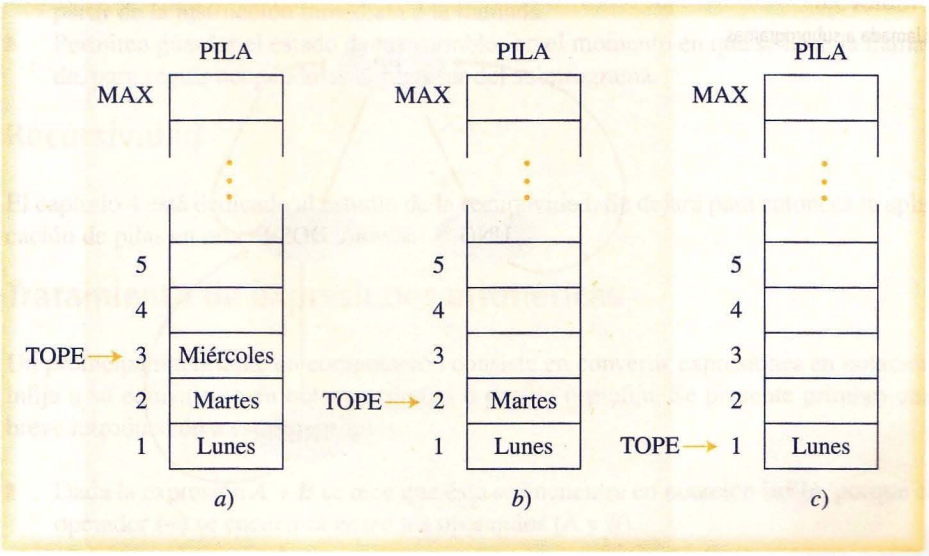


Una forma de resolver este problema es eliminar primeramente los elementos *jueves* y *miércoles*, de esta manera *martes* quedaría ubicado en la cima de PILA y ahora sería posible extraerlo (figuras 3.7*a*, 3.7*b* y 3.7*c*).

3.2.3 Aplicaciones de pilas

Las pilas son una estructura de datos muy usada en la solución de diversos tipos de problemas, en el área de la computación. Ahora se analizarán algunos de los casos más representativos de aplicación de las mismas:

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



- 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 las 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, porque a esa posición se regresa el control del proceso.

Supongamos, por ejemplo, que se tiene un programa principal (*PP*) que llama a los subprogramas UNO y DOS. A su vez, el subprograma DOS llama al TRES. Cada vez que la ejecución de uno de los subprogramas concluye, se regresa el control al nivel inmediato superior (fig. 3.8).

Cuando el programa *PP* llama a UNO, se guarda en una pila la posición en la que se hizo la llamada (fig. 3.9*a*). Al terminar UNO, el control se regresa a *PP* recuperando previamente la dirección de la pila (fig. 3.9*b*). Al llamar a DOS, nuevamente se guarda la dirección de *PP* en la pila (fig. 3.9*c*). Cuando DOS llama a TRES, se pone en la pila la dirección de DOS (fig. 3.9*d*). Después de procesar TRES, se recupera la posición de DOS para continuar con su ejecución (fig. 3.9*e*). Al terminar DOS se regresa el control a *PP*, obteniendo previamente la dirección guardada en la pila (fig. 3.9*f*).

Finalmente podemos concluir que las pilas son necesarias en este tipo de aplicaciones por lo siguiente:

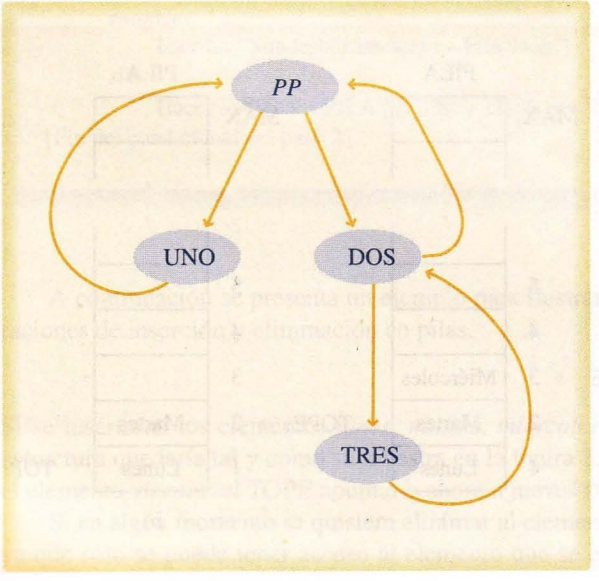
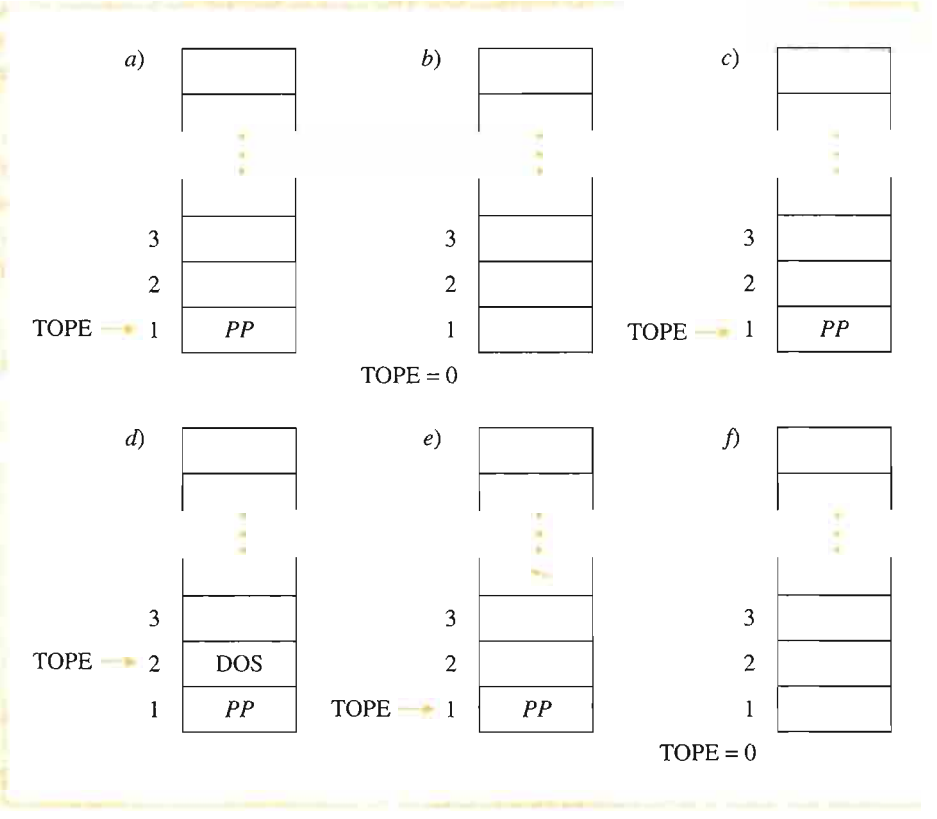


FIGURA 3.8
Llamada a subprogramas.

FIGURA 3.9
Aplicación de pilas: llamadas a subprogramas.



- 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.

Recursividad

El capítulo 4 está dedicado al estudio de la recursividad. Se dejará para entonces la aplicación de pilas en procesos recursivos.

Tratamiento de expresiones aritméticas

Un problema interesante en computación consiste en convertir expresiones en notación infija a su equivalente en notación prefija o posfija o prefija. Se presenta primero una breve introducción a estos conceptos.

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

- Dada la expresión *AB+* se dice que ésta se encuentra en notación **postfija**, porque el operador (+) se encuentra **después** de los operandos (*A* y *B*).
- Dada la expresión *+AB* se dice que ésta se encuentra en notación **prefija**, porque el operador (+) está **precediendo** a los operandos (*A* y *B*).

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

Para convertir una expresión dada en notación infija a una en notación postfija o prefija se establecen primero ciertas condiciones:

- Solamente se manejarán los siguientes operadores —se presentan de mayor a menor según sea su prioridad de ejecución—:

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

- 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 encuentran entre paréntesis tendrán más prioridad que cualquier operador.

Se presentan a continuación, paso a paso, algunos ejemplos de conversión de expresiones infijas a notación posfija.

Ejemplo 3.2

En este ejemplo se exponen dos casos de traducción de notación infija a posfija. El primero de ellos es una expresión simple, mientras que el segundo presenta mayor grado de complejidad. En la tabla 3.1 se muestran los pasos necesarios para lograr la traducción de la primera expresión, y en la tabla 3.2 los correspondientes a la segunda expresión.

a) Expresión infija: *X + Z * W*

Expresión posfija: *XZW*+*

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. Se coloca el operador de

TABLA 3.1
Traducción de infija a posfija

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

tal manera que los operandos afectados por él lo precedan. 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**.

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

Expresión postfija: *XZ+W*TY^/V–*

TABLA 3.2
Traducción de expresión infija a posfija.

Paso	Expresión
0	$(X + Z)^* \; W/T \; \wedge \; Y - V$
1	$XZ + \; *W/T \; \wedge \; Y - V$
2	$XZ + \; *W/TY \; \wedge \; - V$
3	$XZ + W*/TY \; \wedge \; - V$
4	$XZ + W * TY \; \wedge \; / \; - V$
5	$XZ + W * TY \; \wedge \; / \; V -$

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, y así con los demás, según su jerarquía. 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. El operador de la resta es el último que se mueve, paso 5. A continuación se presenta el algoritmo que traduce una expresión infija a otra posfija.

Algoritmo 3.5 Conv_postfija

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

```
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
```

Cabe señalar que para este algoritmo se maneja la escala de prioridades presentada al inicio de esta sección.

Ejemplo 3.3

En este ejemplo se retoman los casos del ejemplo 3.2 para ilustrar el funcionamiento del algoritmo Conv_posfija.

a) Expresión infija: $X + Z * W$

Expresión posfija: $XZW*+$

En la tabla 3.3 se presentan los pasos necesarios para lograr la traducción deseada siguiendo el algoritmo 3.5.

En los pasos 1, 3 y 5 el símbolo analizado —un operando— se agrega directamente a EPOS. Al analizar el operador +, paso 2, se verifica si en PILA hay operadores con 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

TABLA 3.3
Traducción de expresión infija a posfija

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				$XZW * +$

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.

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

Expresión postfija: $XZ+W*TY\wedge/V-$

En la tabla 3.4 se presentan los pasos necesarios para lograr la traducción deseada, siguiendo el algoritmo 3.5.

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 polaca 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.

A continuación se presenta el algoritmo para convertir expresiones infijas a expresiones escritas en notación prefija.

Ejemplo 3.4

En este ejemplo se exponen dos casos de traducción de notación infija a prefija. El primero de ellos es una expresión simple, mientras que el segundo presenta mayor grado de complejidad.

a) Expresión infija: $X + Z * W$

Expresión prefija: $+ X*ZW$

TABLA 3.4
Traducción de expresión infija a 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	$Y - V$	$\wedge$	/ $\wedge$	$XZ+W*T$
11	$- V$	Y	/ $\wedge$	$XZ+W*TY$
		–	/	$XZ+W*TY\wedge$
12	V	–		$XZ+W*TY\wedge/$
		–	–	$XZ+W*TY\wedge/$
13		V	–	$XZ+W*TY\wedge/V$
14				$XZ+W*TY\wedge/V-$

En la tabla 3.5 se presentan los pasos necesarios para lograr la traducción deseada.

Como en el caso de la notación postfija, ejemplo 3.2, aquí también el operador de multiplicación se procesa primero. De la traducción de la expresión, paso 1, resulta el operador precediendo a los operandos. Lo mismo para el operador de suma, paso 2.

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

Expresión prefija: $-/*+XZW\wedge TYV$

En la tabla 3.6 se presentan los pasos necesarios para lograr la traducción deseada.

Lo primero que se procesa en este caso es la subexpresión que se encuentra entre paréntesis, paso 1. El orden en que se procesan los operadores es el mismo que se siguió

TABLA 3.5
Traducción de expresión infija a prefija

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

TABLA 3.6
Traducción de expresión infija a prefija

Paso	Expresión
0	$(X + Z) * W / T \wedge Y - V$
1	$+ XZ * W / T \wedge Y - V$
2	$+ XZ * W / \wedge TY - V$
3	$* + XZW / \wedge TY - V$
4	$/ * + XZW \wedge TY - V$
5	$- / * + XZW \wedge TYV$

para la conversión de infija a posfija. Por tanto, sería reiterativo volver a explicar paso a paso el contenido de la tabla 3.6. Sin embargo, es de destacar la posición que ocupan los operadores con respecto a los operandos: los primeros preceden a los segundos.

A continuación se incluye el algoritmo de conversión de notación infija a prefija. Este algoritmo se diferencia del anterior básicamente en el hecho de que los elementos de la expresión en notación infija se recorrerán de derecha a izquierda.

Algoritmo 3.6 Conv_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

dad; por tanto, permanecerá en PILA hasta el final del proceso de conversión, paso 15. Cuando se encuentra un paréntesis derecho, paso 9, se agrega directamente a PILA, mientras que cuando el símbolo analizado es un paréntesis izquierdo, paso 13, se extrae repetidamente cada elemento de PILA agregándolo a EPRE, hasta que se encuentra un paréntesis derecho. Éste se retira de PILA y no se agrega a EPRE. Cuando ya se analizaron todos los símbolos de la expresión se procede a quitar de PILA sus elementos, añadiéndolos a EPRE. Finalmente se invierte EPRE para obtener la expresión en notación prefija, paso 17. Para evitar el último paso del algoritmo, invertir la expresión, se podrían ir concatenando los símbolos en EPRE en orden inverso.

Ordenación

Otra aplicación de las pilas se puede ver en el método de ordenación rápida. Como el tema de ordenación es ampliamente tratado en el capítulo 8, se sugiere remitirse a él.

3.2.4 La clase *Pila*

La **clase *Pila*** tiene atributos y métodos. Los atributos son la colección de elementos y el TOPE. Los métodos, por otra parte, son todas aquellas operaciones analizadas en la sección anterior —Pila_vacía, Pila_llena, Pone y Quita—. En la figura 3.10 se puede observar gráficamente la clase Pila.

Se tiene acceso a los miembros de un objeto de la clase *Pila* por medio de la notación de puntos. Al asumir que la variable PIOBJ representa un objeto de la clase Pila previamente creado, se puede hacer:

PIOBJ.Pila_llena para invocar el método que determina si la pila está llena o no. En este método no hay argumentos, ya que todos los valores requeridos son miembros de la clase.

PIOBJ.Pone(argumento) para insertar un nuevo elemento en la pila. En este método sólo hay un argumento que indica el valor a guardar en la pila; los demás valores requeridos son miembros de la clase.

FIGURA 3.10
Clase *Pila*.