9.5. EVALUACIÓN DE EXPRESIONES ARITMÉTICAS CON PILAS

Una *expresión aritmética* está formada por operandos y operadores. La expresión `x*y - (a+b)` consta de los operadores `*`, `-`, `+` y de los operandos `x`, `y`, `a`, `b`. Los operandos pueden ser valores constantes, variables o, incluso, otra expresión. Los operadores son los símbolos conocidos de las operaciones matemáticas. La evaluación de una expresión aritmética da lugar a un valor numérico, se realiza sustituyendo los operandos que son variables por valores concretos y ejecutando las operaciones aritméticas representadas por los operadores. Si los operandos de la expresión anterior toman los valores `x = 5, y = 2, a = 3, b = 4`, el resultado de la evaluación es:

```
5*2 - (3+4)   = 5*2 - 7 = 10 - 7 = 3
```

La forma habitual de escribir expresiones matemáticas es aquella en la que el operador está entre sus dos operandos. La expresión anterior está escrita de esa forma, y recibe el nombre de *notación infija*. Esta forma de escribir las expresiones exige, en algunas ocasiones, el uso de paréntesis para *encerrar* subespresiones con mayor prioridad.

Los operadores, como es sabido, tienen distintos niveles de precedencia o prioridad a la hora de su evaluación. A continuación se recuerdan estos niveles de prioridad en orden de mayor a menor:

```
Paréntesis              : ( )
Potencia                :  ^
Multiplicación/división :  *, /
Suma/Resta              :  +, -
```

Normalmente, en una expresión hay operadores con la misma prioridad. A igualdad de precedencia, los operadores se evalúan de izquierda a derecha *(asociatividad)*, excepto la potencia, que es de derecha a izquierda.

9.5.1. Notación prefija y notación postfija de una expresiones aritmética

La forma habitual de escribir operaciones aritméticas *(notación infija)* sitúa el operador entre sus dos operandos. Esta forma de notación obliga, en muchas ocasiones, a utilizar paréntesis para indicar el orden de evaluación. Las expresiones siguientes:

```
r =  a*b/(a+c)      g = a*b/a+c
```

son distintas al no poner paréntesis en la expresión `g`. Igual ocurre con estas otras:

```
r =  (a-b)^c+d      g = a-b^c+d
```

Existen otras formas de escribir expresiones aritméticas, que se diferencian por la ubicación del operador respecto de los operandos. La notación en la cual el operador se coloca delante de los dos operandos, se conoce como *notación prefija* y se conoce como notación polaca (en honor del matemático polaco que la propuso).

Ejemplo 9.5

Dadas las expresiones: `a*b/(a+c); a*b/a+c; (a-b)^c+d` *. Escribir las expresiones equivalentes en notación prefija.*

Se escribe, paso a paso, la transformación de cada expresión aritmética en la expresión equivalente en notación polaca.

```
a*b/(a+c)   (infija) → a*b/+ac → *ab/+ac → /*ab+ac (polaca)
a*b/a+c     (infija) → *ab/a+c → /*aba+c → +/*abac (polaca)
(a-b)^c+d   (infija) → -ab^c+d → ^-abc+d → +^-abcd (polaca)
```

En el Ejemplo 9.4, se observa que no son necesarios los paréntesis al escribir la expresión en notación polaca. Ésta es su propiedad fundamental: el orden de ejecución de las operaciones está determinado por las posiciones de los operadores y los operandos en la expresión.

Notación postfija

Hay más formas de escribir las expresiones. La tercera notación más utilizada se denomina *postfija* o *polaca inversa* y coloca el operador a continuación de sus dos operandos.

Ejemplo 9.6

Dadas las expresiones: `a*b/(a+c); a*b/a+c; (a-b)^c+d`*, escribir las expresiones equivalentes en notación postfija.*

Paso a paso, se transforma cada subexpresión en notación *postfija*.

```
a*b/(a+c)   (infija) → a*b/ac+ → ab*/ac+ → ab*ac+/ (polaca inversa)
a*b/a+c     (infija) → ab*/a+c → ab*a/+c → ab*a/c+ (polaca inversa)
(a-b)^c+d   (infija) → ab-^c+d → ab-c^+d → ab-c^d+ (polaca inversa)
```

> ***Norma***
>
> Las diferentes formas de escribir una misma expresión aritmética dependen de la ubicación de los operadores respecto a los operandos. Es importante tener en cuenta que tanto en la notación prefija como en la postfija no son necesarios los paréntesis para cambiar el orden de evaluación.

9.5.2. Evaluación de una expresión aritmética

La evaluación de una expresión aritmética escrita de *manera habitual*, en *notación infija*, se realiza en dos pasos principales:

1. Transformar la expresión de notación *infija* a *postfija*.
2. Evaluar la expresión en notación *postfija*.

En los algoritmos que se aplican en cada uno de los pasos es básico el uso del *TAD Pila*. La estructura pila asegura que el *último en entrar es el primero en salir* y, de esa forma, el algoritmo de transformación a *postfija* sitúa los operadores después de sus operandos con la prioridad o precedencia que les corresponde. Una vez que se tiene la expresión en notación *postfija,* se utiliza otra pila, de elementos numéricos, para guardar los valores de los operandos y de las operaciones parciales con el fin de obtener el valor numérico de la expresión.

9.5.3. Transformación de una expresión infija a postfija

Se parte de una expresión en notación *infija* que tiene operandos, operadores y puede tener paréntesis. Los operandos vienen representados por letras, mientras los operadores son éstos:

```
^ (potenciación), *, /, +, - .
```

La transformación se realiza utilizando una pila en la que se almacenan los operadores y los paréntesis izquierdos. La expresión aritmética se lee del teclado y se procesa carácter a carácter. Los operandos pasan directamente a formar parte de la expresión en *postfija,* la cual se guarda en un *array.* Un operador se mete en la pila si se cumple que:

- La pila esta vacía.
- El operador tiene mayor prioridad que el operador cima de la pila.
- El operador tiene igual prioridad que el operador cima de la pila y se trata de la máxima prioridad.

Si la prioridad es menor o igual, se saca el elemento cima de la pila, se pone la expresión en *postfija* y se vuelve a hacer la comparación con el nuevo elemento cima.

El paréntesis izquierdo siempre se mete en la pila; ya en la pila, se les considera de mínima prioridad para que todo operador que se encuentra dentro del paréntesis entre en la pila. Cuando se lee un paréntesis derecho, hay que sacar todos los operadores de la pila y ponerlos en la expresión *postfija,* hasta llegar a un paréntesis izquierdo, el cual se elimina, ya que los paréntesis no forman parte de la expresión *postfija.*

El algoritmo termina cuando no hay más ítems de la expresión origen y la pila está vacía. Por ejemplo, la expresión `a*(b+c-(d/e^f)-g)-h` está escrita en *notación infija*, la expresión equivalente en *postfija* se va a ir formando paso a paso:

Expresión en postfija

`a` Operando `a` pasa a la expresión *postfija;* operador `*` a la pila.

`ab` Operador `(` pasa a la pila; operando `b` a la expresión.

`abc` Operador + pasa a la pila; operando `c` a la expresión.

En este punto, el estado de la pila es:

El siguiente carácter de la expresión, -, tiene igual prioridad que el operador de la cima (+) y da lugar a:

`abc+`

`abc+d` El operador `(` se mete en la pila; el operando `d` a la expresión.

`abc+de` El operador `/` pasa a la pila; el operando `e` a la expresión.

`abc+def` El operador `^` pasa a la pila; el operando `f` a la expresión.

El siguiente ítem, `)` (paréntesis derecho), produce que se vacíe la pila hasta un `'('`. La pila, en este momento, dispone de estos operadores:

`abc+def^/` El algoritmo saca operadores de la pila hasta un `'('` y da lugar a la pila:

`abc+def^/-` El operador – pasa a la pila y, a su vez, se extrae –; el siguiente carácter, el operando `g`, pasa a la expresión.

`abc+def^/-g` El siguiente carácter es `')'`, por lo que son extraídos de la pila los operadores hasta un `(`, la pila queda :

`abc+def^/-g-` El siguiente carácter es el operador –, hace que se saque de la pila el operador `*` y se meta en la pila el operador –.

`abc+def^/-g-*` Por último, el operando h pasa directamente a la expresión.

`abc+def^/-g-*h` Fin de entrada, se vacía la pila pasando los operadores a la expresión:

`abc+def^/-g-*h-`

En la descripción realizada se observa que el paréntesis izquierdo tiene la máxima prioridad fuera de la pila; es decir, en la *notación infija*; sin embargo, cuando está dentro de la pila, la prioridad es mínima. De igual forma, para tratar el hecho de que dos, o más, operadores de potenciación se evalúan de derecha a izquierda, este operador tendrá mayor prioridad cuando todavía no esté metido en la pila que cuando esté puesto en ella. Las prioridades son determinadas según la Tabla 9.1.

Tabla 9.1 Prioridades de los operadores considerados

Operador	Prioridad dentro pila	Prioridad fuera pila
^	3	4
*, /	2	2
+,-	1	1
(	0	5

Algoritmo de paso de notación *infija* a *postfija*

Los pasos a seguir para transformar una expresión algebraica de notación infija a postfija son:

1. Obtener caracteres de la expresión y repetir los pasos 2 al 4 para cada carácter.
2. Si es un operando, pasarlo a la expresión *postfija*.
3. Si es operador:
   3.1. Si la pila está vacía, meterlo en la pila. Repetir a partir de 1.
   3.2. Si la pila no está vacía:
   Si la prioridad del operador leído es mayor que la prioridad del operador cima de la pila, meterlo en la pila y repetir a partir de 1.
   Si la prioridad del operador es menor o igual que la prioridad del operador de la cima de la pila, sacar el operador cima de la pila y pasarlo a la expresión postfija, volver a 3.
4. Si es paréntesis derecho:
   4.1. Sacar el operador cima y pasarlo a la expresión postfija.
   4.2. Si el nuevo operador cima es paréntesis izquierdo, suprimir el elemento cima.
   4.3. Si la cima no es paréntesis izquierdo, volver a 4.1.
   4.4. Volver a partir de 1.
5. Si quedan elementos en la pila, pasarlos a la expresión postfija.
6. Fin del algoritmo.

Para codificar este algoritmo es necesario crear una pila en la que se almacenen caracteres. Se utiliza el diseño de pila con lista enlazada del apartado 9.4, que se encuentra en el paquete `TipoPila`. Es necesario, para poner un carácter en la pila, crear una referencia de tipo `Character`; al extraer o leer el elemento `cima`, se realizará la conversión inversa: de tipo `Object` a `Character` y llamando al método `charValue()` se obtiene el correspondiente carácter, que será un operador.

Codificación del algoritmo de transformación a postfija

En la clase `ExprePostfija`, se declaran los métodos para transformar una expresión escrita en *infija* a notación *postfija*. La clase hace de *depósito* de los métodos necesarios para realizar la transformación, todos ellos cualificados con `static`.

El método `postFija()` implementa los pasos del algoritmo de transformación, recibe como argumento una cadena con la expresión inicial, crea una pila y, en un bucle de tantas iteraciones como caracteres, realiza las acciones del algoritmo. Este método define el *array* `char[] expsion` de igual longitud que la expresión original. A este *array* se pasarán los elementos que forman la expresión en *postfija*. Una vez que termina la transformación, el método devuelve una cadena con la expresión escrita en notación *postfija;* la cadena se crea a partir del *array* `expsion: new String(expsion, 0, n)`.

El método `prdadDentro()` determina la prioridad de un operador que está dentro de la pila, y `prdadFuera()` la prioridad de un operador fuera de la pila.

(El código fuente se encuentra en la página web del libro, archivo *ExpresionPostfija*.)

9.5.4. Evaluación de la expresión en notación postfija

Una vez que se tiene la expresión en notación postfija, se evalúa la expresión. En primer lugar, hay que dar valores numéricos a los operandos de la expresión. De nuevo, el algoritmo de evaluación utiliza una pila, en esta ocasión de operandos, es decir, de números reales.

Al describir el algoritmo, `expsion` es la cadena con la expresión *postfija*. El número de elementos es la longitud, `n`, de la cadena. Los pasos a seguir son los siguientes:

1. Examinar `expsion` elemento a elemento: repetir los pasos 2 y 3 para cada elemento.
2. Si el elemento es un operando, meterlo en la pila.
3. Si el elemento es un operador, se simboliza con `&`, entonces:
   - Sacar los dos elementos superiores de la pila, que se denominarán `b` y `a` respectivamente.
   - Evaluar `a & b`, el resultado es `z = a & b`.
   - El resultado `z`, meterlo en la pila. Repetir a partir del paso 1.
4. El resultado de la evaluación de la expresión está en el elemento cima de la pila.
5. Fin del algoritmo.

Codificación de la evaluación de expresión en postfija

La clase `EvalPostfija` agrupa los métodos para implementar el algoritmo, todos ellos se declaran `static`. Realmente, son dos los métodos que se escriben: `valorOprdos()` y `evalua()`. El primero, `valorOprdos()`, simplemente es para dar entrada a los valores de los operandos.

El método `double evalua()` implementa el algoritmo y recibe la cadena con la expresión y el *array* con el valor de cada operando. La pila de números reales, utilizada por el algoritmo, se instancia de la clase `PilaLista`, diseñada de tal forma que sus elementos son de tipo `Object`. Entonces, al meter un operando en la pila será necesario crear instancias de tipo `Double`, y al extraer el elemento `cima`, convertir `Object` a `Double`.

(El código fuente se encuentra en la página web del libro, archivo *EvaluarExpresion*.)

Clase principal

El método `main()` gestiona la petición de la expresión original y las sucesivas llamadas a los métodos para transformar de *infija* a *postfija* y, por último, evaluar la expresión para unos valores concretos de los operandos.

```
import java.io.*;
import evaluarExpresion.*;

class EjemploEvaluaExpresion
{
   public static void main(String [] a)
   {
      double []v = new double[26];
      double resultado;
      BufferedReader entrada = new BufferedReader(
                               new InputStreamReader(System.in));
      String expresion;
      try {
         System.out.print("\nExpresión aritmética: ");
         expresion = entrada.readLine();
            // Conversión de infija a postfija
         expresion = ExprePostfija.postFija(expresion);
         System.out.println("\nExpresión en postfija: " + expresion);
            // Evaluación de la expresión
         EvalPostfija.valorOprdos(expresion, v);  // valor de operandos
         resultado = EvalPostfija.evalua(expresion, v);
         System.out.println("Resultado = " + resultado);
      }
      catch (Exception e)
      {
         System.out.println("\nError en el proceso ... " + e);
         e.getStackTrace();
      }
   }
}
```

RESUMEN

- Una *pila* es una estructura de datos tipo **LIFO** (*last in first out*, último en entrar primero en salir) en la que los datos (todos del mismo tipo) se añaden y se eliminan por el mismo extremo, denominado *cima* de la pila.
- Se definen las siguientes operaciones básicas sobre pilas: `crear`, `insertar`, `cima`, `quitar`, `pilaVacia`, `pilaLlena` y `limpiarPila`.
- `crear` inicializa la pila como pila vacía. Esta operación la implementa el constructor.
- `insertar` añade un elemento en la cima de la pila. Debe haber espacio en la pila.
- `cima` devuelve el elemento que está en la cima, sin extraerlo.
- `quitar` extrae de la pila el elemento cima de la pila.
- `pilaVacia` determina si el estado de la pila es vacía, en cuyo caso devuelve el valor lógico *true*.
- `pilaLlena` determina si existe espacio en la pila para añadir un nuevo elemento. De no haber espacio, devuelve `true`. Esta operación se aplica en la representación de la pila mediante *array*.

- `limpiarPila` el espacio asignado a la pila se libera, quedando disponible.
- Las aplicaciones de las pilas en la programación son numerosas; entre ellas está la evaluación de expresiones aritméticas. Primero se transforma la expresión a notación postfija, a continuación se evalúa.
- Las expresiones en notación polaca, postfija o prefija, tienen la característica de que no necesitan paréntesis.

EJERCICIOS

9.1. ¿Cuál es la salida de este segmento de código, teniendo en cuenta que el tipo de dato de la pila es `int`?

```
Pila p = new Pila();
int x = 4, y;

p.insertar(x);
System.out.println("\n " + p.cimaPila());
y = p.quitar();
p.insertar(32);
p.insertar(p.quitar());
do {
   System.out.println("\n " + p.quitar());
}while (!p.pilaVacia());
```

9.2. Escribir el método `mostarPila()` para escribir los elementos de una pila de cadenas de caracteres, utilizando sólo las operaciones básicas y una pila auxiliar.

9.3. Utilizando una pila de caracteres, transformar la siguiente expresión a su equivalente expresión en postfija.

`(x-y)/(z+w) - (z+y)^x`

9.4. Obtener una secuencia de 10 números reales, guardarlos en un *array* y ponerlos en una pila. Imprimir la secuencia original y, a continuación, imprimir la pila extrayendo los elementos.

9.5. Transformar la expresión aritmética del Ejercicio 9.3 en su expresión equivalente en notación prefija.

9.6. Dada la expresión aritmética `r = x*y - (z+w)/(z+y)^x`, transformar la expresión a notación postfija y, a continuación, evaluar la expresión para los valores: `x = 2, y = -1, z = 3, w = 1`. Para obtener el resultado de la evaluación seguir los pasos del algoritmo descrito en el Apartado 9.5.3.

9.7. Se tiene una lista enlazada a la que se accede por el primer nodo. Escribir un método que imprima los nodos de la lista en orden inverso, desde el último nodo al primero; como estructura auxiliar, utilizar una pila y sus operaciones.

9.8. La implementación del *TAD Pila* con *arrays* establece un tamaño máximo de la pila que se controla con el método `pilaLlena()`. Modificar este método de tal forma que, cuando se llene la pila, se amplíe el tamaño del *array* a justamente el doble de la capacidad actual.