Estructuras de Datos.
Facultad de Ciencias, UNAM 2015-2.
Práctica 06.

Armando Ballinas Nangüelú.
armballinas@gmail.com

Fecha de entrega 14 de abril de 2015

1. Introducción.

En esta práctica se ímplementará el TDA *pila* para realizar una calculadora sencilla de expresiones aritméticas. Estas expresiones estarán en notación posfija[1] y para evaluarlas será necesario el uso de una pila.

2. El tipo de dato abstracto *pila*.

Una pila es una estructura de datos lineal que satisface la propiedad en inglés L.I.F.O (Last In, First Out). Lo cual significa que el último elemento en ser añadido será el primero en ser retirado.

Una pila permite operaciones básicas parecidas a las listas. Se puede calcular su longitud, insertar, borrar e iterar. En las pilas la operación de insertar se conoce como *apilar* o en inglés *push* y la operación de eliminar se conoce como *desapilar* o en inglés *pop*.

La operación *pop* devuelve el tope de la pila eliminándolo de la misma. Si solo se quiere conocer el tope pero no borrarlo se debe implementar la operación *peek*, la cual solo devuelve el tope de la pila pero no lo elimina.

2.1. Implementación de una pila.

Una pila se puede implementar a partir de una lista. Simplemente los elementos de la pila se almacenan en la lista y para insertar un nuevo elemento se inserta al inicio de la lista.

Lo anterior implica que para eliminar un elemento se debe eliminar el elemento al inicio de la lista. Lo anterior satisface la condición L.I.F.O. de ls pilas.

3. Notación posfija.

La manera usual de escribir las operaciones aritméticas es en notación infija, es decir, el operador está enmedio de los operandos, así escribimos una suma de la siguiente forma: `4 + 5`. Esta manera

[1] A veces conocida como "polaca".

hace que sea más fácil leer y entender las operaciones, sin embargo tiene desventajas. Una de ella es que debemos aprender que los operadores tienen una característica llamada *precedencia*. Así, la multiplicación se realiza antes que la suma por lo que evaluar `4 + 5 * 2` da como resultado `14` y no `18`. Sin embargo, sabemos que la multiplicación y la división tienen la misma precedencia, por lo que para poder evaluar `5 * 4 / 2 * 3` es necesario el uso de paréntesis.

La notación posfija, es una notación en donde el operador va al final de la misma. Por ejemplo, la suma del ejemplo anterior se escribe `4 5 +` y para la última expresión, si los paréntesis son `(5 * 4) / (2 * 3)` entonces en notación posfija quedaría así: `5 4 * 2 3 */`. Una ventaja de la notación posfija es que vuelve inecesaria la precedencia y también los paréntesis. La principal desventaja es que es más difícil leerla.

3.1. Transformación de una expresión a notación posfija

Para transformar una expresión de notación infija a posfija se debe utilizar una pila. Los elementos de la pila serán los elementos o *tokens* de la expresión infija original más un *token* adicional que diga que hemos terminado, digamos #. Los *tokens* están formados por los números , los operadores `+,-,*,/` y los paréntesis `(,)`.

El primer paso del proceso es transformar una cadena de una expresión en notación infija a una lista de *tokens*. Para esto se utilizará una clase adicional.

Una vez que tengamos la lista de tokens, llamémosle I, $I[i]$ denotará el *token* en la posición i. El siguiente algoritmo transforma una lista de *tokens* en notación infija a una lista de *tokens* en notación posfija utilizando una pila S.

En el algoritmo se supone existe una función que determina si un *token* es un operando, es decir un número, o no. Además se debe conocer la precedencia de los operadores. El paréntesis `(` tiene precedencia 3. Los operadores `*,/` tienen 2. Los operadores `+,-,)` tienen 1 y el operador de fin de expresión `#` tiene 0.

4. Evaluación de una expresión en notación posfija.

Para evaluar una expresión en notación posfija se usa otra pila. El procedimiento es el siguiente:

1. Leer *token* por *token* la expresión.
2. Si el *token* actual es un operando, se apila.
3. Si no lo es, entonces es un operador y se realiza la operación con el tope de la pila y con el siguiente en la pila los cuales se desapilan. El resultado se apila.
4. El proceso termina cuando termina la expresión. La pila aloja al resultado final.

5. Ejercicios.

1. (3 pts.) Realizar la implementación del TDA Pila. Las operaciones que deben implementar son: *push*, *pop*, *peek*, *length* e *isEmpty*.
2. (5 pts.) Realizar la implementación de una clase `Evaluate` que evalúe una expresión aritmética. Primero debe transformar la expresión aritmética a notación posfija y luego evaluarla en notación posfija. Esta clase debe almacenar la expresión en notación posfija.

Algoritmo 1 Transformación de notación infija a posfija

Entrada L es una lista de *tokens* en notación infija.

Salida Una lista de *tokens* P en notación posfija.

```
1: function CONVIERTE(L)
2:     S ← ∅
3:     i ← 0
4:     j ← 0
5:     S.push('#')
6:     while S ≠ ∅ do
7:         if esOperando(L[i]) then
8:             P[j] ← L[i]
9:             j++
10:            i++
11:        else
12:            if S.peek() = '(' then
13:                if L[i] = ')' then
14:                    i++
15:                    S.pop()
16:                else
17:                    S.push(L[i])
18:                    i++
19:                end if
20:            else
21:                if prioridad(S.peek()) < prioridad(L[i]) then
22:                    S.push(L[i])
23:                    i++
24:                else
25:                    P[j] ← S.pop()
26:                    j++
27:                end if
28:            end if
29:        end if
30:    end while
31: end function
```

3. (2 pts.) Realizar una clase `Main` que sirva como prueba de la práctica. Esta clase debe leer por línea de comandos (como argumento) una cadena que será la expresión en notación infija a evaluar y deberá imprimir la expresión equivalente en notación posfija así como su evaluación.