PILA

Es una colección ordenada de elementos en la cual en un extremo se pueden insertar y retirar elementos; este extremo es llamado parte superior de la pila (tope).

PILA DE SEIS ELEMENTOS

[F]
[E]
[D]
[C]
[B]
[A]

El elemento más alto es f, d es más alto que c, b y a, pero menor que e y f. Una pila se utiliza para cálculos o para ejecución de instrucciones, puede implantarse por hardware de diseño especial o en memoria controladas por programas.

Según la definición de una pila un solo extremo de la pila puede designarse como parte superior, los nuevos elementos que se inserten deben colocarse en la parte superior de la pila, en éste caso la pila se mueve hacia arriba. Cuando los elementos son removidos o eliminados la pila se mueve hacia abajo. Se debe tener un indicador (apuntador) del tope de la pila pata llevar a cabo los movimientos.
PUSH es la función que ejecuta la inserción de elementos, POP retira elementos del tope de la pila.

OPERACIONES BASICAS DE LAS PILAS

s.Push(x) = insertar un elemento x en la pila **s** (stack).
s.Pop()= remover un elemento de la pila **s**.

PILA VACÍA

Una pila se denomina vacía cuando ésta no contiene elementos. Por lo tanto antes de retirar un elemento es necesario asegurarse que la pila no esté vacía, esto será con la operación **s.isEmpty()**, si ésta regresa el valor de verdadero la pila **s** está vacía de lo contrario retorna un valor de falso.

Otra operación puede ser determinar el elemento superior de la pila sin retirarlo, esto será con la operación **s.Stacktop()**.

Es necesario utilizar siempre **s.isEmpty()** ya que debemos asegurarnos que la pila está vacía de lo contrario si pedimos que nos diga cual es el elemento superior (**s.Stacktop()**) nos marcaría un error de *underflow*.

APLICACIONES DE PILAS

Supongamos que se da una expresión matemática que incluye varios conjuntos de paréntesis y se desea saber si los paréntesis están embebidos correctamente es decir:

1. Si existe igual número de paréntesis a la izquierda que a la derecha.
2. Cada paréntesis de la derecha esté precedido por el correspondiente paréntesis de la izquierda.

Entonces podrá decirse que cada paréntesis a la derecha cierra una posibilidad. En algún punto particular de la expresión la cantidad de paréntesis abiertos y que no han sido cerrados se le conocerá como **profundidad de embebimiento.**

CONTADOR DE PARÉNTESIS.-

En un punto particular de la expresión, el número de paréntesis a la izquierda menos el número de paréntesis a la derecha.

Ejemplo:

```
7 - ( ( X * ( ( X + Y ) / ( J - 3 ) ) + Y ) / ( 4 - 2 . 5 ) )
0  1 2     3 4        3   4      3  2      1   2           1 0
```

NOTA: para que no exista profundidad de embebimiento el resultado del contenido del contador debe ser igual a cero.

En el caso anterior usamos sólo un tipo de paréntesis, sin embargo podemos asumir que existen tres tipos de separadores diferentes {},(),[]. Cualquier tipo se separador que se encuentra al final debe ser igual al del principio. Por consiguiente es necesario tener en cuenta no solamente cuántos separadores han sido abiertos sino también de que tipo son.

Una pila se puede utilizar para registrar diferentes tipos de signos de agrupación; en cualquier momento que se encuentre un signo abriendo la expresión éste es empujado hacia la pila y cada vez que se encuentre el correspondiente signo terminal se examina la pila, si la pila está vacía quiere decir que el signo de agrupación terminal no tiene correspondiente de apertura. Si la pila no está vacía retiramos un elemento y revisamos si corresponde al signo de agrupación terminal, si coincide continuamos si no es así, la hilera no es válida, cuando llegamos al final de la hilera nos aseguramos de que la pila esté vacía, de lo contrario quiere decir que se han abierto uno o más signos pero no se han cerrado, haciendo que la hilera no sea válida.

ALGORITMO DE VERIFICACION DE PARENTESIS

```
1. Hacer valid=1
2. Hacer stack=0
3. Si ya se ha leído toda la cadena o valid=0
      ir a paso #5
   de lo contrario
3a) lee symb "{" siguiente de la cadena
         3b) si symb= "(" o symb= "[" o symb= "{" entonces
```

```
                s.Push(symb) ir a paso #3
         3c)  si symb= ")" o symb= "]" o symb= "}" entonces
                si s.isEmpty() entonces
                     valid=0
                 de lo contrario
                      X= s.Pop()
        Si X <> simbolo de apertura equivalente a symb entonces vali=0
4. Ir a paso # 3
5. Si no s.isEmpty() entonces
           Valid=0
6. si valid=1 entonces
            escribe "cadena válida"
       de lo contrario
             escribe "cadena no válida"
7. Fin.
```

CALCULADORA POSTFIJA.-

Considérese la suma A y B, pensando que la aplicación del operador "+" a los operandos A y B, se escribirá A+B, esta representación se denomina infijo o entrefijo.

Existen otras dos formas de representar la suma de A + B utilizando el signo "+", estas son el prefijo y el postfijo. Los prefijos PRE y POST se refieren a la posición relativa del operador con respecto a los operandos.

En la notación de PREFIJO el operador precede a los dos operandos y en la notación POSTFIJO el operador va después de los dos operandos.

Consideremos en la expresión a+b*c escrita en infijo, observe que existen dos operadores, se quiere saber cual de estos se realizará primero, es lógico que por jerarquía se realizaría primero la multiplicación y posteriormente la suma.

La única regla durante el proceso de conversión es que las operaciones que tienen relación o jerarquía más alta se convertirá primero y después de que esta parte de la expresión ha sido convertida se maneja como operando simple.

El orden de precedencia o jerarquía de los operadores son: primero el exponente, posteriormente multiplicación y división y por último la suma y la resta, Cuando dos operadores tienen la misma precedencia, se sigue el orden de izquierda a derecha, a excepción de la exponenciación donde se asume un orden de derecha a izquierda.

REGLAS

1.- Dar jerarquías o precedencias sin afectar a la expresión
2.- De acuerdo a las jerarquías iniciar las conversiones
3.- Las conversiones se harán de la siguiente manera:

    a) El orden de los operandos no cambia, solo el de los operadores

b) Los paréntesis que aparezcan el la expresión original, no aparecerán en las conversiones

c) El operador irá precedido de los operandos y operadores que afecten, esto en el caso del prefijo. En forma contraria funcionará el postfijo.

En expresiones de postfijo no existen los paréntesis, pero el orden en que se presentan los operadores determina el orden actual de las operaciones al evaluar la expresión, haciendo en este caso que los paréntesis no sean necesarios.

Al hacer las expresiones de infijo a postfijo a simple vista perdemos la habilidad de observar a los operandos asociados con un operador, por lo tanto la notación postfija de la expresión original puede parecer simple si no fuera por el hecho que parece difícil de evaluar. La solución a este problema es la aplicación de un algoritmo para evaluar expresiones en postfijo.

ALGORITMO PARA EVALUAR EXPRESIONES EN POSTFIJO

```
1.- Si no hay más caracteres en la cadena entonces
       ir al paso # 5
    de lo contrario
2.-    Lee symb
3.- Si symb = operandos entonces
       opndstk.Push(symb)
    de lo contrario
       opnd2= opndstk.Pop()
       opnd1= opndstk.Pop()
       valor=resultado de aplicar
       symb a opnd1 sobre opnd2
       opndstk.Push(valor)
4.- ir al paso # 1
5.- Resultado = opndstk.Pop()
```

Donde:

opndstk = PILA DE OPERANDOS
opnd2 = PRIMER OPERANDO RETIRADO
opnd1 = SEGUNDO OPERANDO RETIRADO
symb = SIGUIENTE SIMBOLO A LEER

Ejemplo: 3-(2*8)+9 = 328*-9+ = -4