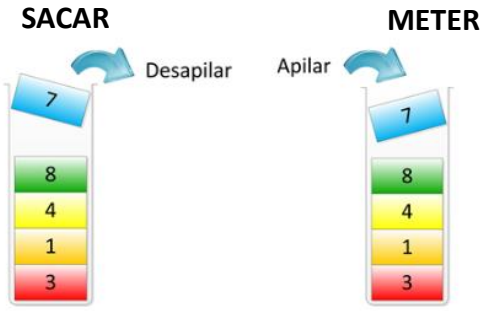
UNIDAD VI TDA Pila

2.1.1 Descripción del TDA Pila

Una pila es un tipo de lista en el que todas las inserciones y eliminaciones de elementos se realizan por el mismo extremo de la lista.

El nombre de pila procede de la similitud en el manejo de esta estructura de datos y la de una “pila de objetos". Estas estructuras también son llamadas listas ***LIFO (LAST IN, FIRST OUT)*** acrónimo que refleja la característica más importante de las pilas.

Es fácil encontrar ejemplos de pilas en la vida real: hay una pila en el aparato dispensador de platos de un autoservicio, o en el cargador de una pistola automática, o bien en una calle sin salida. En todos estos casos, el ultimo ítem que se añade a la pila es el primero en salir.



2.1.2 Especificación del TDA Pila

Partiendo de lo establecido en la Unidad 1. Especificación Informal tenemos lo siguiente:
Elementos que conforman la estructura

TDA Pila (**VALORES** todos los valores numéricos de cualquier tipo, **OPERACIONES** crear, vacía, cima, meter, sacar)

OPERACIONES

crear (P: Pila)
Utilidad: Sirve para inicializar la pila
Entrada: Pila P
Salida: Ninguna
Precondición: Ninguna.
Poscondición: Pila inicializada

Vacía (P: Pila)
Utilidad: Sirve para indicar si la pila esta vacia
Entrada: Pila P
Salida: Valor booleano
Precondición: Ninguna.
Poscondición: Ninguna

Cima (P: Pila)
Utilidad: Devuelve el elemento situado en la cima de la pila
Entrada: Pila P

Salida: Elemento
Precondición: La pila es no vacía.
Poscondición: Ninguna

Meter (ES P: Pila, E : Elemento)
Utilidad: Añade el elemento E a la pila, quedando éste situado en el tope
Entrada: Pila P y Elemento E
Salida: Ninguna
Precondición: Pila con capacidad.
Poscondición: Pila modificada con un elemento más en su contenido

Sacar (ES P: Pila,ES E: Elemento)
Utilidad: Suprime el elemento situado en el tope de la pila y lo retorna
Entrada: Pila P y receptor de elemento E
Salida: ninguna, el valor del elemento retorna en el parámetro por referencia E
Precondición: Pila no vacia.
Poscondición: Pila modificada con un elemento menos en su contenido

2.1.3 Aplicaciones con Pila

En esta sección se puede apreciar que ya estamos en condiciones para plantear algoritmos usando el TDA Pila, abstrayéndonos de la forma como esta implementado.

Evaluar expresiones

Para poder evaluar expresiones tenemos que tomar en cuenta que existen dos formatos de expresión muy importantes que, al principio, pueden no parecer obvios. Considere la expresión **infija A + B.** ¿Qué pasaría si moviéramos el operador antes de los dos operandos? La expresión resultante sería **+ A B.** Del mismo modo, podríamos mover el operador al final de la expresión y obtendríamos **A B +**. Estas expresiones se ven un poco extrañas.

Estos cambios en la posición del operador con respecto a los operandos crean dos nuevos formatos de expresión, la **notación prefija** y la **notación sufija (o postfija)**. La notación prefija requiere que todos los operadores precedan a los dos operandos sobre los que actúan. La notación sufija, por otro lado, requiere que sus operadores aparezcan después de los operandos correspondientes. Algunos ejemplos más deberían ayudar a hacer esto un poco más claro según el siguiente gráfico:

Expresión infija	Expresión prefija	Expresión sufija
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +
(A + B) * C	* + A B C	A B + C *
A + B * C + D	+ + A * B C D	A B C * + D +
(A + B) * (C + D)	* + A B + C D	A B + C D + *
A * B + C * D	+ * A B * C D	A B * C D * +
A + B + C + D	+ + + A B C D	A B + C + D +

Aplicación 1: Evaluación de expresiones postfijas
Dada la siguiente expresión: 2+3*5 → 235*+ =17

Real EvalurarPostfija (ExpPostfija: Cadena)
INICIO
// Llamar al constructor de la pila P
Para cada i= 1 hasta longitud(ExpPostfija) hacer // iniciamos recorrido en la cadena
INICIO
SI ExpPostfija[i] esta en ('0','1','2','3','4','5','6','7','8','9') // Si es operando metemos en pila
ENTONCES
p.meter(ExpPostfija[i])
CASO CONTRARIO // Si es un operador sacamos los dos primeros elementos
INICIO
p.sacar(Op2) // de la pila y realizamos la operación indicada con
p.sacar(Op1) // ellos. El resultado lo volvemos a meter en la pila
sím_operacion= ExpPostfija[i]
Z=Evalua(Op1, sim_operacion , Op2)
p.meter(Z)
FIN
FIN
// Al final la pila queda con un único elemento que es el resultado de la evaluación
EvaluarPostfija = p.cima()
FIN

Real Evalua(Op1 :real; Operador:char; Op2:real)
INICIO
SI OPERADOR= '^' ENTONCES Evalua =exp(op2*ln(op1)) FIN SI
SI OPERADOR='*' ENTONCES Evalua =Op1*Op2 FIN SI
SI OPERADOR='/' ENTONCES Evalua =Op1*Op2 FIN SI
SI OPERADOR='+' ENTONCES Evalua =Op1*Op2 FIN SI
SI OPERADOR='-' ENTONCES Evalua =Op1*Op2 FIN SI
FIN

Aplicación 2. Conversión de notación infija a postfija
Dada la siguiente expresión: 2+3*5 → 235*+

```
Cadena InfijaToPostFija(Infija: Cadena)
INICIO
// Llamar al constructor de la pila P
Postfija=''
PARA CADA i=1 to longitud(infija) hacer   // Recorremos la expresión infija
   INICIO
    SI infija[i] esta en ('0','1','2','3','4','5','6','7','8','9')  //Operando pasa a la expresión postfija
       ENTONCES Postfija=Postfija + Infija[i]
       CASO CONTRARIO
            SI Infija[i] está en  ('^','*','/','+','-','(')  // Si es un operador
                 ENTONCES
                    INICIO
                      salir=falso
                      MIENTRAS NO( SALIR )  HACER
                      INICIO // Compara prio del operador de la expresión con ultimo guardado
                       Aux=p.cima()
                      SI (p.Vacia)  o  (PrioridadInfija(Infija[i])>PrioridadPila(aux)
                          ENTONCES  INICIO
                                      p.meter(Infija[i]) // se mete el operador en la pila
                                      salir=verdad
                                    FIN
                          CASO CONTRARIO
                                    INICIO //Si es< o = sacar el operador de la pila a la expresión
                                    p.sacar(Aux)
                                    Postfija=Postfija + aux
                                    FIN
                      FIN // fin mientras
                FIN
              CASO CONTRARIO            // Si encontramos un paréntesis derecha se sacan
                 SI Infija[i]=')' ENTONCES // operadores de la pila hasta encontrar un
                              REPETIR      // paréntesis izquierdo que se elimina.
                                 p.sacar(aux)
                                 SI aux<> '('  ENTONCES
                                       Postfija=postfija + aux
                              HASTA QUE aux='('
FIN  // Cuando se acaba la expresión infija se vacía la pila en la expresión postfija
MIENTRAS NO p.vacia HACER
     INICIO
       p.sacar(aux)
      postfija=postfija + aux
     FIN
InfijaToPostfija=Postfija
FIN
....
```

```
Entero PrioridadInfija (c: caracter)
INICIO
SI C='^' ENTONCES prioridadInfija =4 FIN SI
SI C='*' ENTONCES prioridadInfija =2 FIN SI
SI C='/' ENTONCES prioridadInfija =2 FIN SI
SI C='+' ENTONCES prioridadInfija =1 FIN SI
SI C='-' ENTONCES prioridadInfija =1 FIN SI
SI C='(' ENTONCES prioridadInfija =5 FIN SI
FIN
```

```
Entero Prioridadpila (c: caracter)
INICIO
SI C='^' ENTONCES prioridadpila=3 FIN SI
SI C='*' ENTONCES prioridadpila=2 FIN SI
SI C='/' ENTONCES prioridadpila =2 FIN SI
SI C='+' ENTONCES prioridadpila =1 FIN SI
SI C='-' ENTONCES prioridadpila =1 FIN SI
SI C='(' ENTONCES prioridadpila =0 FIN SI
FIN
```

Aplicación 3. Análisis de paréntesis en una expresión
Verificar si la expresión tiene correctamente colocados los paréntesis.

```
Booleano ParentesisOk (Expresion : Cadena)
INICIO
   // LLAMAR CONSTRUTOR DE PILA P
  PARA CADA i = 1 hasta longitud(expresión) hacer
      INICIO
          SI expresión[i]='(' entonces
                                  p.meter(expresion[i])
          SI expresion[i]=')' entonces
                                  p.sacar(aux)
      FIN
     ParentesisOk= (p.vacia)
FIN
```

2.1.4 Implementaciones del TDA Pila

En esta sección mostraremos tres implementaciones para el TDA Pila:

- **Implementación con lista**
- **Implementación con Vectores**
- **Implementación con Simulación de Memoria/Punteros**

2.1.4.1 Implementación con lista.

```
Clase Pila
Atributos
     L : Lista
Metodos
    Crear()
  booleano Vacia()
  Meter(E : TipoElemento)
  Sacar( ES E: TipoElemento)
  Tipoelemento cima()
Fin
Constructor Pila.Crear inicio
    // Crear Objeto L
fin
Pila.meter( E: TipoElemento)
  Inicio
  l.inserta(l.primero(),E)
```

```
 fin
pila.Sacar ( es E : TipoElemento)
inicio
    l.recupera(l.primero(),E)
    l.suprime(l.primero)
fin
booleano pila.vacia()
inicio
   retornar l.vacia
fin
Tipoelemento pila.cima()
inicio
  l.recupera(l.primero,E)
 retornar E
fin
```

2.1.4.2 Implementación con vectores

Dirección de tipo Entero

Clase Pila

Atributos

elementos[Max] vector de tipo TipoElemento

Tope de tipo Direccion

Metodos

```
   Crear()
  booleano Vacia()
  Meter(E : TipoElemento)
  Sacar( ES E: TipoElemento)
  Tipoelemento cima()
```

Fin

```
Pila.Crear
  inicio
   tope = 0
Fin
Booelano Pila.vacia()
inicio
retornar (tope = 0)
Fin
Pila.meter ( E : TipoElemento)
inicio
   si tope<MAX entonces
   tope= tope +1
   elementos[ tope ] = E
fin
Pila.Sacar ( ES E: TipoElemento )
inicio
si no vacia() entonces
     e= elementos[ tope ]
     tope = tope - 1
```

```
      caso contrario // Error
Fin
TipoElemento pila.cima()
Inicio
si no vacia() entonces
    retornar elementos[ tope ]
fin
```

2.1.4.3 Implementación con simulación de Memoria

```
Nodo
        elemento TipoElemento
        sig     Puntero a Nodo
// fin definicion

 Direccion Puntero a espacio de memoria de tipo Nodo
 Clase Pila
        Atributos
            Tope Puntero de tipo Direccion
        Metodos
           Crear()
          booleano Vacia()
          Meter(E : TipoElemento)
          Sacar( ES E: TipoElemento)
          Tipoelemento cima()
        Fin
Constructor crear
Inicio
  tope=-1
fin

Pila.meter( E : TipoElemento)
inicio
aux = // Pedir Espacio de memoria para Nodo
si aux <> nulo entoces
            ponerdato(aux,'->elemento',E)
            ponerdato(aux,'->sig',Tope)
            Tope = Aux
           caso contrario // Error
fin
TipoElemento Pila.cima()
  inicio
    Si(vacia()) Entoces  // Error
     caso contrario
        return obtenerdato(Tope,'->elemento')
fin
Pila.sacar(ES E:TipoElemento)
  inicio
    Si(vacia()) Entoces  // Error
```

```
        caso contrario
           x=tope
           E=obtenerdato(Tope,'->elemento')
           Tope=obtener_dato(tope,'->sig')
           Delete_espacio(x)
fin
```