Implementación del TAD Lista

La implementación dinámica más sencilla para representar los valores del TAD lista es mediante un único puntero apuntando al nodo que contiene el primer elemento de la lista, y donde cada nodo está compuesto por el elemento y un puntero apuntando al siguiente nodo de la lista hasta el último nodo, que apunta a *nulo*.

```
tipos
    ptUnDato = puntero a unDato;
    unDato = registro
                dato: elemento;
                sig: ptUnDato;
             fregistro

    lista = ptUnDato;
```

El coste en tiempo de las operaciones, en el peor de los casos, es:

operación	coste
crearLista	O(1)
añadeIzq	O(1)
creaUnitaria	O(1)
eliminaIzq	O(1)
observaIzq	O(1)
esVacía	O(1)
concatena	O(long (l))
añadeDch	O(long (l))
observaDch	O(long (l))
long	O(long (l))
está	O(long (l))
inser	O(long (l))
borra	O(long (l))
modif	O(long (l))
observa	O(long (l))
pos	O(long (l))

Si ampliamos la estructura del tipo, añadiendo un puntero hacia al último elemento de la lista y un contador que conserve el número de elementos, algunas de las operaciones pasan a ser de orden constante

```
tipos   ptUnDato = puntero a unDato;
        lista = registro
                    prim, ult: ptUnDato;
                    n: entero;
                fregistro

        unDato =  registro
                      dato: elemento;
                      sig: ptUnDato;
                  fregistro
```

operación	coste
concatena	O(1)
añadeDch	O(1)
observaDch	O(1)
long	O(1)

```
módulo listas
importa defTipoElemento
exporta
    tipo lista

    acción creaVacía (var l: lista)
    acción añadeIzq (e: elemento; var l: lista)
    acción concatena (var l1: lista; l2: lista)
    acción creaUnitaria (e: elemento; var l: lista)
    acción añadeDch (var l: lista: e: elemento)
    acción eliminaIzq (var l: lista)
    acción eliminaDch (var l: lista)
    función observaIzq (l: lista): elemento
    función observaDch (l: lista): elemento
    función long (l: lista): entero
    función está (e: elemento; l: lista): booleano
    función esVacía (l: lista): booleano
    acción inser (var l: lista; i: entero; e: elemento)
    acción borra (var l: lista; i: entero)
    acción modif (var l: lista; i: entero; e: elemento)
    función observa (l: lista; i: entero): elemento
    función pos (e: elemento; l: lista): entero
    acción asignar (var nueva: lista; vieja: lista)
    acción liberar (var l: lista)
```

```
implementación
  tipos   tUnDato = puntero a unDato;
          lista = registro
                    prim, ult: ptUnDato;
                    n: entero;
                  fregistro

          unDato = registro
                     dato: elemento;
                     sig: ptUnDato;
                   fregistro
```

```
  acción creaVacía (var l: lista)

     l.prim:= nulo;
     l.ult:= nulo;
     l.n:= 0;
  facción
```

```
  acción añadeIzq (e: elemento; var l: lista)
  var
       aux: ptUnDato
  fvar

     reservar (aux);
     aux^.dato:= e;
     aux^.sig:= l.prim;
     l.prim:= aux;
     si l.ult = nulo entonces
        l.ult:= l.prim;
     fsi
     l.n:= l.n +1;
  facción
```

```
  acción concatena (var l1: lista; l2: lista)

     si l2.n ≠ 0 entonces
        si l1.n = 0 entonces
           l1.prim:= l2.prim;
           l1.ult:= l2.ult;
           l1.n:= l2.n;
        sino
           l1.ult^.sig:= l2.prim;
           l1.ult:= l2.ult;
           l1.n:= l1.n + l2.n
        fsi
     fsi
  facción
```

```
acción creaUnitaria (e: elemento; var l: lista)

    reservar (l.prim);
    l.prim^.dato:= e;
    l.prim^.sig:= nulo;
    l.ult:= l.prim;
    l.n:= 1;
facción


función observaIzq (l: lista): elemento

    retorna (l.prim^.dato)
ffunción


función observaDch (l: lista): elemento

    retorna (l.ult^.dato)
ffunción


función long (l: lista): entero

    retorna (l.n)
ffunción


acción añadeDch (var l: lista: e: elemento)

    si l.ult = nulo entonces
        creaUnitaria (e, l)
    sino
        reservar (l.ult^.sig);
        l.ult:= l.ult^.sig;
        l.ult^.dato:= e;
        l.ult^.sig:= nulo;
        l.n:= l.n + 1;
    fsi
facción


función esVacía (l: lista): booleano

    retorna (l.n = 0)
ffunción
```

```
acción eliminaIzq (var l: lista)
var
    aux: ptUnDato;
fvar

    aux:= l.prim;
    l.prim:= l.prim^.sig;
    si l.prim = nulo entonces
      l.ult:= nulo;
    fsi
    l.n:= l.n – 1;
    liberar (aux);
facción
```

```
acción eliminaDch (var l: lista)
var
    aux: ptUnDato
fvar

    si l.prim = l.ult entonces
      liberar (l.prim);
      creaVacía (l);
    sino
      aux:= l.prim
      mientras aux^.sig ≠ l.ult hacer
        aux:= aux^.sig;
      fmientras
      aux^.sig:= nulo;
      liberar (l.ult);
      l.ult:= aux;
      l.n:= l.n – 1;
    fsi
facción
```

```
función está (e: elemento; l: lista): booleano
var
    aux: ptUnDato;
    encontrado: booleano;
fvar

    si l.prim = nulo entonces
      retorna (falso);
    sino
      aux:= l.prim;
      mientras (aux^.dato ≠ e) and (aux ≠ l.ult) hacer
        aux:= aux^.sig;
      fmientras
      retorna (aux^.dato = e)
    fsi
ffunción
```

```
acción inser (var l: lista; i: entero; e: elemento)
var
    aux, sigAux: ptUnDato;
    indSig: entero;
fvar

    en caso de
        i = 1: añadeIzq (e, l);
        i = long (l): añadeDch (l, e);
    en otros casos
        aux:= l.prim;
        indSig:= 2;
        mientras indSig < i hacer
          aux:= aux^.sig;
          indSig:= indSig + 1;
        fmientras
        sigAux:= aux^.sig;
        reservar (aux^.sig);
        aux:= aux^.sig;
        aux^.dato:= e;
        aux^.sig:= sigAux;
        l.n:= l.n + 1;
    fcaso
facción
```

```
acción borra (var l: lista; i: entero)
var
    aux, sigAux: ptUnDato;
    indSig: entero;
fvar

    en caso de
        i = 1: eliminaIzq (l);
        i = long (l): eliminaDch (l);
    en otros casos
        aux:= l.prim;
        indSig:= 2;
        mientras indSig < i hacer
          aux:= aux^.sig;
          indSig:= indSig + 1;
        fmientras
        sigAux:= aux^.sig;
        aux^.sig:= sigAux^.sig;
        liberar (sigAux);
        l.n:= l.n - 1;
    fcaso
facción
```

```
acción modif (var l: lista; i: entero; e: elemento)
var
    aux: ptUnDato;
    ind: entero;
fvar

    si i = long (l) entonces
        l.ult^.dato:= e;
    sino
        aux:= l.prim;
        ind:= 1;
        mientras ind < i hacer
          aux:= aux^.sig;
          ind:= ind + 1;
        fmientras
        aux^.dato:= e;
    fsi
facción
```

```
función observa (l: lista; i: entero): elemento
var
    aux: ptUnDato;
    ind: entero;
fvar

    si i = long (l) entonces
        retorna (l.ult^.dato);
    sino
        aux:= l.prim;
        ind:= 1;
        mientras ind < i hacer
          aux:= aux^.sig;
          ind:= ind + 1;
        fmientras
        retorna (aux^.dato);
    fsi
ffunción
```

```
función pos (e: elemento; l: lista): entero
var
    aux: ptUnDato;
    i: entero;
fvar

    aux:= l.prim;
    i:= 1;
    mientras aux^.dato ≠ e hacer
      aux:= aux^.sig;
      i:= i + 1;
    fmientras
    retorna (i);
ffunción
```

```
acción asignar (var nueva: lista; vieja: lista)
var
    i: entero
fvar

    creaVacía (nueva);
    para i:= 1 hasta long (vieja) hacer
        añadeDch (nueva, vieja.prim^.dato);
        vieja.prim:= vieja.prim^.sig;
    fpara
facción


acción liberar (var l: lista)
var
    n, i: entero;
fvar

    n:= long (l);
    para i:= 1 hasta n hacer
        eliminaIzq (l);
    fpara
facción


fmódulo
```