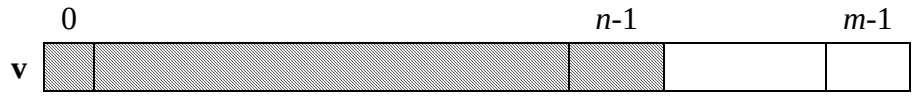
# Representaciones Contiguas (I)

## • Contigua lineal:

(*m* = capacidad del vector, *n* = número de elementos almacenados)



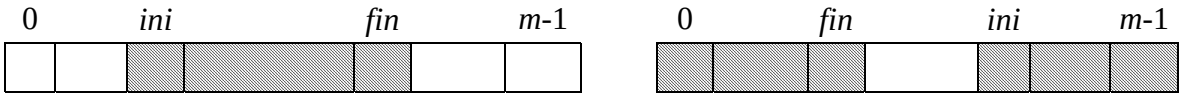

- Acceso a elemento posición *i* : v[i]
- Inserción de elemento *x* en posición *i* : ($0 \le i \le n$)

```
if n = m then ampliar(v);
for j := n-1 downto i do v[j+1] := v[j];
v[i] := x;
n := n+1
```

- Borrado de elemento en posición *i* : ($0 \le i < n$)

```
for j := i+1 to n-1 do v[j-1] := v[j];
n := n-1
```

## • Contigua circular:




- Acceso a elemento posición *i*: v[(i+ini-1) **mod** m]
- Inserción de elemento *x* en posición *i* : ($0 \le i \le n$)

```
if n = m-1 then ampliar(v);
p = (ini+i) mod m; { posición i-ésimo }
j := fin; { se desplazan n-i elementos }
for k := 1 to n-i do { j va de fin a p }
begin
   v[(j+1) mod m] := v[j]; { v_{j+1} := v_j }
   j := (j+m-1) mod m { j := j-1 }
end;
v[p] := x;
fin := (fin+1) mod m; { fin := fin+1 }
n := n+1
```

**Desplazamiento de la mitad derecha**

```
if n = m-1 then ampliar(v);
p = (ini+i) mod m; { posición i-ésimo }
j := ini; { se desplazan i elementos }
for k := 1 to i do { j va de ini a p }
begin
   v[(j+m-1) mod m] := v[j]; { v_{j-1} := v_j }
   j := (j+1) mod m; { j := j+1 }
end;
v[p] := x;
ini := (ini+m-1) mod m; { ini := ini-1 }
n := n+1
```

**Desplazamiento de la mitad izquierda**

# Representaciones Contiguas (II)

- Borrado de elemento en posición $i$ : $(0 \le i < n)$

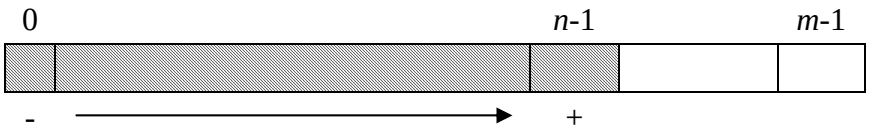
```
p = (i+ini) mod m; { posición i-ésimo }
j := (p+1) mod m ; {desplazar n-i+1 elem}
for k := 1 to n-i-1 do { j va de p+1 a fin }
begin
   v[(j+m-1) mod m] := v[j]; { v_{j-1} := v_j }
   j := (j+1) mod m; { j := j+1 }
end;
fin := (fin+m-1) mod m; { fin := fin-1 }
n := n-1
```

**Desplazamiento de la mitad derecha**

```
p = (i+ini) mod m; { posición i-ésimo }
j := (p+m-1) mod m; {desplazar i-1 elem}
for k := 1 to i-1 do { j va de p-1 a ini }
begin
   v[(j+1) mod m] := v[j]; { v_{j+1} := v_j }
   j := (j+m-1) mod m; { j := j-1 }
end;
ini := (ini+1) mod m; { ini := ini+1 }
n := n-1
```

**Desplazamiento de la mitad izquierda**

- **Contigua ordenada**:




- Acceso a elemento posición $i$ ($i$-ésimo menor): v[i]
- Inserción de elemento $x$ :

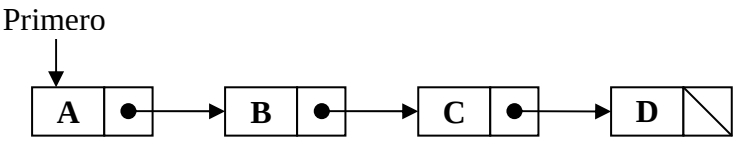
```
if n = m then ampliar(v);
{ Búsqueda binaria de posición donde insertar }
izda := 0; dcha := n-1;
while izda <= dcha do
begin
   { Invariante: v[0..izda-1] <= x, v[dcha+1..n-1] > x }
   med := (izda+dcha) div 2;
   if v[med] <= x then izda := med+1 else dcha := med-1
end;
{ Post: 0 <= izda <= n, v[0..izda-1] <= x < v[izda..n-1] }
for j := n-1 downto izda do v[j+1] := v[j];
v[izda] := x;
n := n+1
```

- Borrado de elemento en posición $i$ : $(0 \le i < n)$

```
for j := i+1 to n-1 do v[i-1] := v[i];
n := n-1
```

# Representaciones Enlazadas (I)

- Ejemplo:




- Enlaces mediante índices:

```
const N = …
type
   TNodo = record
      dato : char; sig : integer;
   end;
   TVector = array[1..N] of TNodo;
var
   V : TVector; primero : integer;
begin
   primero := 3;
   V[3].dato := 'A'; V[3].sig := 5;
   V[5].dato := 'B'; V[5].sig := 1;
   V[1].dato := 'C'; V[1].sig := 4;
   V[4].dato := 'D'; V[4].sig := -1;
end;
```

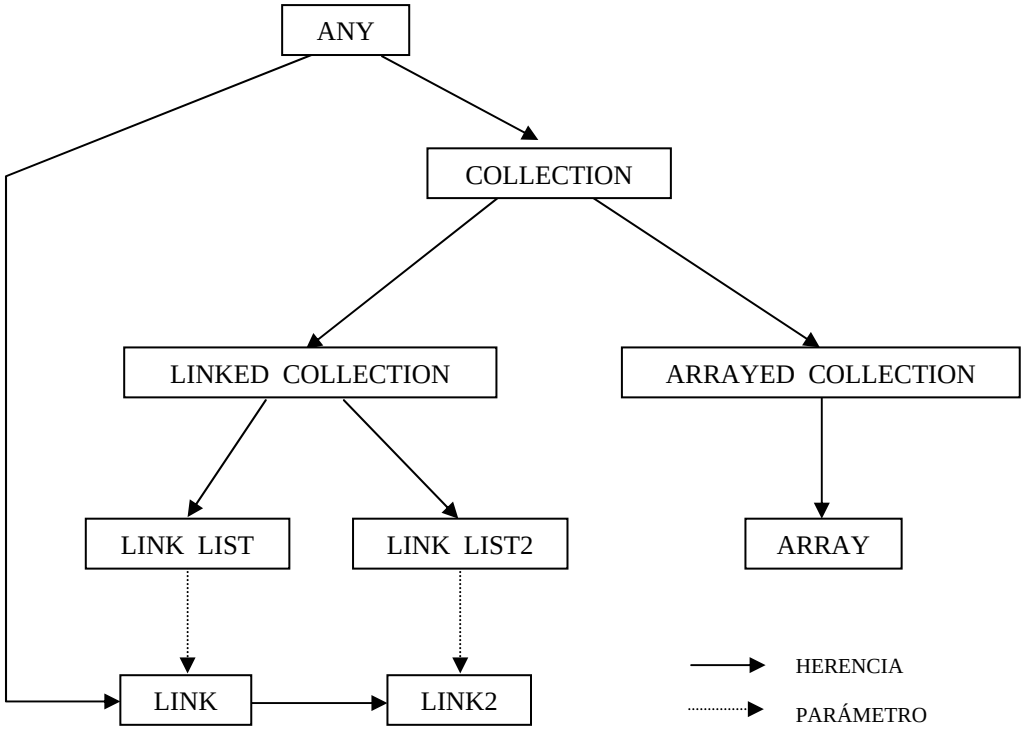
| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| **C** | | **A** | **D** | **B** | |
| **4** | | **5** | **-1** | **1** | |

- Enlaces mediante referencias a objetos:

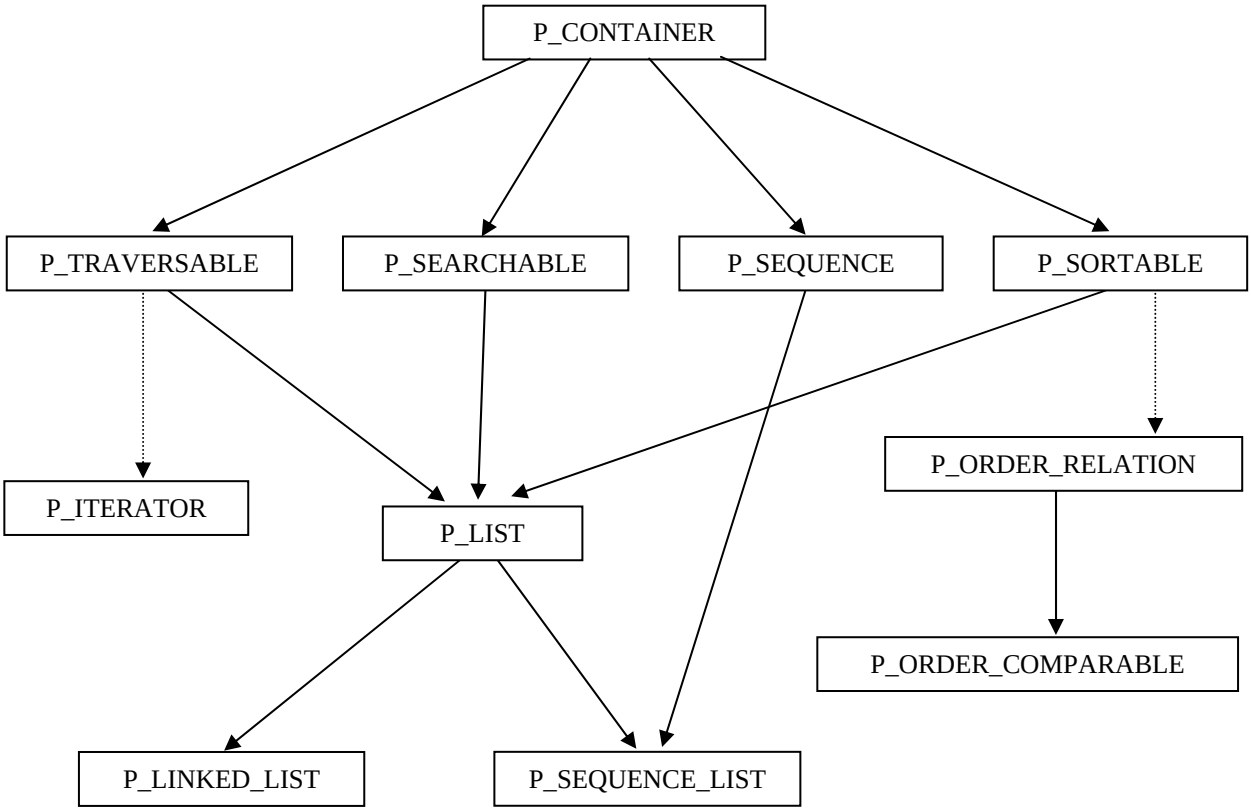
```
class NODO[E]
creation make
feature
   dato : E;
   sig : like Current;

   make(e: like elem; s: like sig) is
   do
      dato := e; sig := s;
   end;

end -- NODO
```

```
class EJEMPLO
creation make
feature
   primero, temp : NODO[CHARACTER];

   make is
   do
      create temp.make('D',Void);
      create temp.make('C',temp);
      create temp.make('B',temp);
      create temp.make('A',temp);
      primero = temp
   end
end -- EJEMPLO
```

# Listas en SmallEiffel




# Listas en Pylon



# Ejemplo de implementación de lista basada en cursor

```
indexing
  "Nodo simplemente enlazado que"
  "almacena un valor de tipo ELEMENTO"
class NODO[ELEMENTO]
creation make
feature { LISTA_CUR }
    elem : ELEMENTO;
    sig : like Current;

    make(e: like elem; s: like sig) is
    do
        elem := e; sig := s
    end

    cambia_sig(s: like sig) is
    do
        sig := s
    end
end -- class NODO
```

```
indexing
    descripcion : "Definición del TAD LISTA_CUR."
                  "Clase abstracta salvo la operación concatenar"
deferred class LISTA_CUR[ELEMENTO]
feature { ANY }

    crear_lc is deferred
    ensure -- Vacía la lista (cursor fuera de lista)
    end -- crear_lc

    fin_de_lista : BOOLEAN is deferred
    ensure  -- True si el cursor está situado despues del último elemento
            -- (fuera de lista) o la lista está vacía. False en caso contrario.
    end -- fin_de_lista

    actual : ELEMENTO is
    require not fin_de_lista
    deferred
    ensure  -- Devuelve el elemento sobre el que está situado el cursor
    end -- actual

    ir_a_inicial is deferred
    ensure -- El cursor se situa sobre el primer elemento. Si la lista está
           -- vacía el cursor sigue fuera de lista.
    end -- ir_a_inicial

    ir_a_siguiente is deferred
    ensure -- El cursor pasa al siguiente elemento. Si el cursor estaba
           -- sobre el último elemento o fuera de lista pasa a estar
           -- fuera de lista.
end -- ir_a_siguiente

    insertar(elem: ELEMENTO) is deferred
    ensure -- inserta el elemento antes del actual. El cursor no cambia. Si
           -- el cursor estaba fuera de lista se inserta el elemento al final
           -- y el cursor sigue fuera de lista.
    end -- insertar

    cambiar(elem: ELEMENTO) is
    require not fin_de_lista
    deferred
    ensure -- cambia el elemento actual y el cursor pasa al siguiente
    end -- cambiar

    quitar is
    require not fin_de_lista
    deferred
    ensure -- borra elemento actual, cursor pasa a siguiente
    end -- quitar

    -- Concatena la otra lista al final de ésta. El cursor de ambas
    -- listas pasa a estar fuera de lista. No se modifica el contenido
    -- de la otra lista (se comparten referencias a elementos).
    concatenar(otra : like Current) is
    do
      -- situar cursor fuera de lista
      from until fin_de_lista loop
        ir_a_siguiente
      end
      -- recorrer la otra lista insertando sus elementos
      from otra.ir_a_inicio until otra.fin_de_lista loop
          insertar(otra.actual)
          otra.ir_a_siguiente
      end
    end -- concatenar
end -- class LISTA_CUR
```

# Implementaciónes contigua y enlazada

```
indexing
    descripcion : "Implementación de la clase"
        "abstracta (TAD) LISTA_CUR"
    representacion :  "Contigua lineal con un índice al"
        "elemento sobre  el que está situado el cursor"
    eficiencia : "Inserción peor caso O(n), amortizado"
        "O(n-i), borrado orden O(n-i), concatenación"
        "O(n+m), resto operaciones O(1). n es el"
        "número de elementos, i la posición del cursor"
        "m nº de elementos de la lista que se concatena"
class LISTA_CUR_CL[ELEMENTO]
inherit LISTA_CUR[ELEMENTO]
creation crear_lc
feature { LISTA_CUR_CL }
    vec : ARRAY[ELEMENTO]
    num : INTEGER -- Numero de elementos
    act : INTEGER -- Indice del elemento actual (cursor)
feature { ANY }
    crear_lc is
    do
      create vec.make(1,100)
      num := 0 ; act := 1
    end -- crear_lc
    fin_de_lista : BOOLEAN is
    do
      Result := act > num
    end -- fin_de_lista
    actual : ELEMENTO is
    do
      Result := vec @ act
    end -- actual
    ir_a_inicial is
    do
      act := 1
    end -- ir_a_inicial
    ir_a_siguiente is
    do
      act := act+1
    end -- ir_a_siguiente

    -- Continua en la siguiente página
```

```
indexing
    descripcion : "Implementación de la clase"
        "abstracta (TAD) LISTA_CUR"
    representacion : "Enlazada simple circular con referencia"
        "adicional al elemento ANTERIOR al cursor y una"
        "indicación de cursor fuera de lista. Los elementos se"
        "almacenan en objetos cuya clase es NODO"
    eficiencia : "Todas las operaciones son de orden O(1)."
class LISTA_CUR_ESC[ELEMENTO]
inherit
    LISTA_CUR[ELEMENTO]
        redefine concatenar
    end
creation crear_lc
feature { LISTA_CUR_ESC }
    ult : NODO[ELEMENTO] -- Ultimo nodo de la lista
    ant : NODO[ELEMENTO] -- Nodo anterior al nodo actual
    -- Cuando la referencia al anterior apunta al último existe
    -- la doble posibilidad de que el cursor sea el primero
    -- (en una lista circular el primero es el siguiente al
    -- último) o bien que el cursor esté fuera de la lista
    -- (despues del último, esto es necesario para poder
    -- insertar un elemento al final).
    -- Esta variable sirve para indicar cual de esas dos
    -- interpretaciones es la adecuada si ant = ult.
    fin : BOOLEAN
feature { ANY }
    crear_lc is
    do
      ult := Void ; ant := Void
      -- Si la lista está vacía el cursor está fuera de lista
      fin := True
    end -- crear_lc
    fin_de_lista : BOOLEAN is
    do
      Result := fin
    end -- fin_de_lista
    actual : ELEMENTO is
    do
      Result := ant.sig.elem
    end -- actual
    ir_a_inicial is
    do
      ant := ult
      -- Si la lista está vacía el cursor está fuera de lista
      fin := (ult = Void)
    end -- ir_a_inicial
    ir_a_siguiente is
    do
      -- Si la lista vacía o cursor fuera de lista no hace nada
      if ult /= Void and not fin_de_lista then
        ant := ant.sig
        -- Si el cursor estaba en el último elemento pasa a
        -- estar fuera de lista
        fin := (ant = ult)
      end
    end -- ir_a_siguiente

    -- Continua en la siguiente página
```

# Implementaciones (continuación)

```
insertar(elem: ELEMENTO) is
    local i : INTEGER
    do
      -- comprobar capacidad del vector
      if num >= vec.upper then
        vec.resize(1,2*vec.upper)
      end
      -- desplazar actual y posteriores a la derecha
      from i := num until i < act loop
        vec.put(vec @ i, i+1)
        i := i-1
      end
      -- insertar elemento, actualizar actual y número
      vec.put(elem, act)
      act := act+1 -- actual debe ser el mismo
      num := num+1
    end -- insertar

    cambiar(elem: ELEMENTO) is
    do
      vec.put(elem, act)
      act := act+1 -– se pasa al siguiente
    end -- cambiar

    quitar is
    local i : INTEGER
    do
      -- desplazar posteriores al actual a la izquierda
      from i := act+1 until i > num loop
        vec.put(vec @ i, i-1)
        i := i+1
      end
      num := num-1
    end -- quitar

    -- concatenar tal como se define en LISTA_CUR
end -- class LISTA_CUR_CL
```

```
    insertar(elem: ELEMENTO) is
    local nuevo : NODO[ELEMENTO] -- nuevo nodo
    do
      create nuevo.make(elem, Void);
      if ult = Void then -- caso lista vacía
        nuevo.cambia_sig(nuevo) -- nuevo es primero y último
        ult := nuevo ; ant := nuevo
        fin := True -- cursor fuera de lista en este caso
      else
        nuevo.cambia_sig(ant.sig)
        ant.cambia_sig(nuevo)
        -- si se inserta al final se debe adaptar ult y ant
        if fin_de_lista then ult := nuevo; ant := nuevo end
      end
    end -- insertar

    cambiar(elem: ELEMENTO) is
    do
      ant.sig.cambia_elem(elem)
      ir_a_siguiente –– se pasa al siguiente
    end -- cambiar

    quitar is
    do
      if ant.sig = ult then -- borrado del último
        ult := ant
      end
      ant.cambia_sig(ant.sig.sig)
    end -- quitar

    -- Versión especial de concatenar adaptada a esta
    -- implementación. La otra lista se vacía (para evitar el
    -- riesgo de tener estructuras con nodos comunes)
    concatenar(otra: like Current) is
    require
      otra /= Current  -- No se puede concatenar una lista
                       -- consigo misma
    local primero : NODO[ELEMENTO]
    do
      if otra.ult /= Void then -- la otra lista no está vacía
        if ult = Void then -- esta lista vacía
          ult := otra.ult
        else -- ninguna lista vacía
          primero := ult.sig
          ult.cambia_sig(otra.ult.sig)
          otra.ult.cambia_sig(primero)
          ult := otra.ult
        end
      end
      -- hacer que el cursor este fuera de la lista
      ant := ult
      fin := True
      -- vaciar la otra lista
      otra.crear_lc
    end -- concatenar

  invariant
    -- invariante de clase
    fin_de_lista_consistente: (fin = True) implies (ant = ult)
end -- class LISTA_CUR_ESC
```

# Ejemplo de implementación de Pila y Cola

```
indexing
    descripcion : "Implementación del TAD PILA"
    representacion : "Contigua circular"
    eficiencia : "Todas las operaciones O(1) (tpo amort)"
class PILA[ELEM]
creation crear_pila
feature { NONE }
    vec : ARRAY[ELEMENTO];
    tam,num : INTEGER; -- capacidad y numero de elem.

feature { ANY }
    crear_pila is
    do
      tam := 100; num := 0;
      create vec.make(1,tam);
    end -- crear_pila

    vacia : BOOLEAN is
    do
      Result := (num = 0)
    end -- vacia

    cabeza : ELEM is
    require not vacia
    do
      Result := vec @ num
    end -- cabeza

    -- insertar por la cabeza
    insertar(e: ELEM) is
    local i, j : INTEGER
    do
      num := num+1
      if num >= tam then -- Ampliar el vector
        vec.resize(1,2*tam)
      end
      -- insertar al final
      vec.put(e , num)
    end -- insertar

    -- quitar elemento cabeza
    quitar is
    require not vacia
    do
      num := num-1
    end – quitar
end -- class PILA
```

```
indexing
    descripcion : "Implementación del TAD COLA"
    representacion : "Contigua circular"
    eficiencia : "Todas las operaciones O(1) (tpo amortizado)"
class COLA[ELEM]
creation crear_cola
feature { NONE }
    vec : ARRAY[ELEMENTO];
    tam,num : INTEGER; -- capacidad y numero de elementos
    ini,fin : INTEGER; -- límites de zona ocupada

feature { ANY }

    crear_cola is
    do
      tam := 100; num := 0; ini := 0; fin := -1;
      create vec.make(0,tam-1);
    end -- crear_cola

    vacia : BOOLEAN is
    do
      Result := (num = 0)
    end -- vacia

    cabeza : ELEM is
    require not vacia
    do
      Result := vec @ ini
    end -- cabeza

    -- insertar por la cola
    insertar(e: ELEM) is
    local i, j : INTEGER
    do
      num := num+1
      if num >= tam then -- Ampliar el vector
        vec.resize(0,2*tam-1)
        if ini > fin then -- Reorganizar vector
          from i := tam-1; j := 2*tam-1 until i < ini loop
            vec.put(vec @ i , j)
            i := i-1; j := j-1
          end
          ini := j+1
        end
        tam := 2*tam-1
      end
      fin := (fin+1) \\ tam;
      vec.put(e , fin)
    end -- insertar

    -- quitar elemento cabeza
    quitar is
    require not vacia
    do
      num := num-1
      ini := (ini+1) \\ tam
    end -- quitar
end -- class COLA
```