Módulo III

Listas

Una Lista es una secuencia finita de cero o más elementos. Si bien en un sentido amplio los elementos de una lista no tienen por qué ser todos del mismo tipo, nos ocuparemos exclusivamente de listas cuyos elementos son todos del mismo tipo y que en adelante identificaremos como TipoElemento.

Podemos definir a la lista L como una secuencia de cero o más elementos, todos de TipoElemento. Por lo tanto el modelo matemático que la representa será la sucesión $<a_1, a_2,...,a_n>$, $n \geq 0$ donde cada a_i es de TipoElemento.

Cada elemento de una lista tiene asociado un único valor que llamaremos posición.

En una lista los elementos están ordenados linealmente, y es posible, tanto ubicar un determinado elemento en la lista conociendo su posición o "ubicación" en la lista, como saber en qué posición está un elemento en particular.

La posición de un elemento en la lista L puede definirse de distintas maneras, pero en todos los casos debe cumplir lo siguiente: (@)

- Todo elemento de la lista tiene asociada una única posición.
- Dada una posición puede accederse al elemento correspondiente
- Se considerarán (n+1) posiciones válidas para la lista L = $<a_1, a_2,...,a_n>$, $n \geq 0$. Se trata de n+1 posiciones diferentes. De las n+1 posiciones válidas n posiciones corresponden a los n elementos de la lista y hay una posición más que es la posición del elemento a_{n+1} (a_{n+1} no está en la lista L, pero aparecería si agregáramos un elemento más a la lista L).

El concepto de posición puede definirse por ejemplo como un número entero no negativo, de la siguiente manera: posición del elemento a_i es el número i. De esta manera para la lista L = $<a_1, a_2,...,a_n>$, $n \geq 0$ el conjunto de posiciones válidas será {1, 2, ..., n+1}. Observemos que 1,2,...,n son posiciones de elementos de L y que n+1 sería la posición del último elemento de la lista L si le agregáramos un elemento más a la mencionada lista (o sea el elemento a_{n+1}).

elemento	posición válida para L
a_1	1
a_2	2
:	:
a_n	n
a_{n+1} (no existe en L)	n+1

El concepto de posición de un elemento x en la lista L puede definirse de otras maneras, como se mostrará más adelante, siempre que la definición cumpla con los ítems propuestos en (@). En cada caso se busca la definición de posición que más conviene, y una vez determinada, se trabaja de acuerdo a dicha definición.

El TDA: Lista

Para definir un TDA debemos dar el modelo matemático que lo sustenta y un conjunto de operaciones, determinando con precisión el alcance de cada una de ellas.

En el caso del TDA Lista, el modelo matemático es la sucesión $<a_1, a_2,...,a_n>$ con $n \geq 0$ y donde cada a_i es de TipoElemento.

Un conjunto de operaciones para el TDA Lista contendrá, entre otras, operaciones para crear una lista vacía, para actualizar una lista agregando o eliminando elementos en una determinada posición, para conocer el elemento que está en una posición dada y para conocer la posición asociada a un determinado elemento. Un conjunto razonable de operaciones para el TDA Lista es presentado en [Aho83].

Sea L de tipo Lista, x de tipo TipoElemento y p de tipo Posición, definimos las siguientes operaciones:

Fin(L). Esta operación entrega el valor de la posición siguiente a la posición del último elemento. Esta operación es válida para cualquier lista que ha sido creada, tenga o no elementos.

Primera(L). Esta operación devuelve la primer posición de la lista L. Si la lista no es vacía la posición corresponde al primer elemento de la lista y si la lista es vacía la posición primera es Fin(L).

CrearListaVacia (L). Esta operación deja L= < >. Observemos que como L es una lista vacía, la posición Fin(L) coincide con la primera posición de L.

Siguiente(p, L). Esta operación devuelve el valor de la posición que sigue a la posición p en la lista L. Es decir para $L = <a_1, a_2,...,a_n>$, si p es una de las posiciones $1 \leq p \leq n-1$ devolverá p+1, y si p es p_n devolverá Fin(L).
El valor de la operación será indefinido si p no es una posición válida de la lista o si p es Fin(L).

Anterior(p, L). Esta operación devuelve el valor de la posición que precede a la posición p en la lista L, Es decir para $L = <a_1, a_2,...,a_n>$, si p es una de las posiciones $2 \leq p \leq n$ devolverá p-1, y si p es Fin(L) y L no es una lista vacía, devolverá n.
El valor de la operación será indefinido si p no es una posición válida de la lista o si p es Primera(L).

Insertar(x,p,L). Esta operación inserta el elemento x en la posición p de la lista L. Es decir para $L = <a_1, a_2,...,a_{p-1}, a_p,...a_n >$, y $p \neq$Fin(L), luego de Insertar(x,p,L), quedará $L= <a_1, a_2,...,a_{p-1}, x, a_p,...,a_n >$ (el elemento x queda en la posición p, el a_p en la posición siguiente de p y los elementos $a_1, a_2,..., a_{p-1}$, mantienen asociada la misma

posición que tenían antes de la inserción). Para el caso L = <a_1, a_2,...,a_n > y p = Fin(L), luego de Insertar(x,p,L) quedará L = <a_1, a_2,...,a_n ,x> .
El resultado de esta operación es indefinido si la posición p no es una de las n+1 posiciones válidas para L.

Eliminar(p, L). Esta operación suprime el elemento que está en la posición p de la lista L. Es decir, si L= <a_1, a_2,...,a_{p-1}, a_p, a_{p+1},...,a_n > , luego de ejecutar Eliminar(p,L) quedará L= <a_1, a_2,..., a_{p-1}, a_{p+1},..., a_n >. En este caso, los elementos a_1, a_2,..., a_{p-1}, mantienen asociada la misma posición que tenían antes de realizarse la eliminación.
Si p no es una posición válida en L, o p = Fin(L), el resultado de la operación es indefinido.

Recuperar(p, L). Esta operación devuelve el elemento que está en la posición p de L.
El resultado será indefinido si p no es una posición válida de L o si p = Fin(L).

Localizar(x, L). Esta operación devuelve la posición del elemento x en L. Si x está más de una vez en L, la operación devuelve la posición de la "primera aparición de x en L" y si x no es un elemento de L la operación retorna el valor Fin(L).

ListaVacia (L). Esta operación devolverá el valor verdadero si la lista L no tiene elementos y falso en caso contrario. Observemos que una lista está vacía si y solo si cumple la condición Fin(L) = Primer(L).

Imprime(L). Si L = <a_1, a_2,...,a_n>, esta operación imprime los elementos: a_1, a_2,...,a_n en ese orden.

Ejemplos usando el TDA Lista

1- Encontrar la posición del elemento mínimo de una lista L

Si el elemento mínimo de la lista aparece en más de una oportunidad en dicha lista, se tomará la posición de la primer ocurrencia.

```
PROCEDIMIENTO Minimo (L: Lista) :  posicion
  Si ListaVacia(L) entonces
                    ImprimirCartel ( 'la lista está vacía')
                    Retornar (Fin(L))

  Sino
      PosMin:=Primer(L)
      PosActual:=Siguiente(PosMin,L)
      Mientras PosActual <> Fin(L)
          Si Menor (Recuperar (PosActual,L), Recuperar(PosMin,L))
          Entonces      PosMin:=PosActual
          PosActual:=Siguiente(PosActual,L)
      Retornar (PosMin)
FIN Minimo
```

Observemos como se puede plantear un procedimiento en término de los tipos Lista y Posición (independientemente de la implementación de cada uno de ellos)

2- Intercalar dos listas ordenadas en una única lista ordenada.

Sean L1 y L2 las listas ordenadas de entrada y sea ListaResultado la lista donde se alojará el resultado de la intercalación. Consideremos además las posiciones p1 y p2 que señalarán en cada momento los elementos que se están considerando en L1 y L2 respectivamente. La posición p3 indicará en cada momento la posición donde se deberá insertar en la ListaResultado.

Observemos como se puede plantear un procedimiento en término de los tipos Lista, posición y TipoElemento (independientemente de la implementación de cada uno de ellos)

```
PROCEDIMIENTO Intercalar (L1, L2: Lista
                          Var L3: Lista)
p1, p2, p3: Posicion
X1, X2:TipoElemento

(*----------------------------Inicializar---------------------------*)
CrearListaVacia(L3)
p1:= Primera(L1)
p2 := Primera(L2)
p3 := Primera(L3)

(*--------------Intercalación propiamente dicha---------------*)
Mientras p1<>Fin(L1)  y  p2<>Fin(L2)
    X1 := Recuperar(p1,L1)
    X2 := Recuperar(p2,L2)
    Si Menor(X1, X2) entonces  Insertar(X1, p3, L3)
                               p1 := Siguiente(p1,L1)

    Sino si Menor(X2, X1)      Insertar(X2, p3, L3)
                               p2 := Siguiente(p2, L2)

     Sino (* X2=X1 *)          Insertar(X1, P3, L3)
                               p1 := Siguiente(p1, L1)
                               p2 := Siguiente(p2, L2)

    p3 := Siguiente(p3, L3)

(*-------------------------Copiar resto-----------------------------*)
Si p1 <> Fin(L1)  entonces     CopiarResto(p1, L1, p3, L3)

Si p2 <> Fin(L2) entonces      CopiarResto(p2, L2, p3, L3)

FIN Intercalar
```

```
PROCEDIMIENTO  Copiar Resto  (PosFuente: Posicion
                              LtaFuente: Lista
                              PosDestino: Posicion
                              var LtaDestino: Lista)

 Mientras PosFuente <> Fin(LtaFuente) hacer
   Insertar( Recuperar(PosFuente, LtaFuente), PosDestino, LtaDestino)
   PosFuente := Siguiente(PosFuente, LtaFuente)
   PosDestino := Siguiente(PosDestino, LtaDestino )

FIN CopiarResto
```

Implementación del TDA Lista

1- Usando arreglos

Se trata de una representación muy fácil de implementar, apropiada en el caso de manejar listas con pocos elementos y sobre todo cuando no serán frecuentes las inserciones o eliminaciones de elementos al comienzo o en la mitad de la lista.

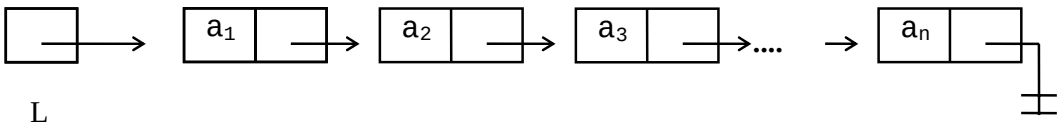
La implementación está totalmente desarrollada en [Aho83], pág. 41.

2- Usando enlaces - Primera versión

Buscando por un lado, una representación que permita manejar listas sin tener que considerar una cota superior al número máximo de elementos que contendrá la lista, y por otro, una estructura que permita hacer inserciones y eliminaciones en cualquier posición de la lista en un tiempo constante, es natural pensar en una estructura dinámica y enlazada. Para ello veremos primero una estructura enlazada y luego dos formas distintas de implementarla (usando punteros y usando cursores)[Señ95].

La forma más natural de implementar listas usando enlaces es mediante el uso de celdas donde cada una de ellas contiene un elemento de la lista y un apuntador a la próxima celda.

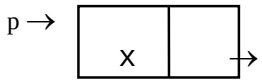
Gráficamente podemos representarla de la siguiente manera:



En el caso de esta representación para acceder a los elementos de la lista basta con tener el valor del enlace L.

El concepto de posición de un elemento x conviene definirlo en este caso como un enlace. Lo definiremos primeramente como el enlace que apunta a la celda que

contiene a dicho elemento. Esta definición permite que en tiempo constante pueda accederse al elemento si se conoce su posición.



Observemos que en esta representación para el caso de L = <a_1, a_2,...,a_n>, se cuenta con n enlaces no nulos (apuntando a cada una de las células que contienen elementos). Como para una lista de n elementos es necesario contar con n+1 posiciones (n correspondientes a los elementos y una más para Fin(L)), teniendo en cuenta la definición de posición que se ha presentado, se adopta por convención que Fin(L) tomará el valor nulo cualquiera sea la lista L.
En esta representación una lista vacía L estará representada por un enlace nulo.

- La definición de la estructura se podrá hacer de la siguiente manera:

```
Posicion = enlace a TipoCelda;
Lista = Posicion;
TipoCelda =  registro con
                elemento: TipoElemento;
                sig: Posicion
```

Observemos que quedan definidos los tipos Lista y Posición. TipoElemento será definido según el problema que se esté resolviendo. Si la implementación se realiza en Modula-2, habrá un módulo ElementoLista donde se define TipoElemento y se importará para la definición de Lista.

Implementación de las operaciones: Se presenta a continuación el análisis y el pseudo-código de alguna de ellas. El resto queda propuesto como ejercicio.

```
PROCEDIMIENTO  CrearListaVacia ( var L:Lista)
│L := nulo
FIN CrearListaVacia

PROCEDIMIENTO Fin (L: Lista) : Posicion
│Retornar (nulo)
FIN Fin

PROCEDIMIENTO Primera (L: Lista) : Posicion
│Retornar (L)
FIN Primera
```

Observar que Primera devuelve la posición del primer elemento si la lista no es vacía, y entrega el valor de Fin en caso contrario.

```
PROCEDIMIENTO Siguiente (p: Posicion; L: Lista) : Posicion
│Si p = nulo entonces Error('Posicion no valida')
│Sino Retornar (Celda(p).sig)
FIN Siguiente
```

Para la operación Insertar observemos que debemos considerar dos casos particulares:

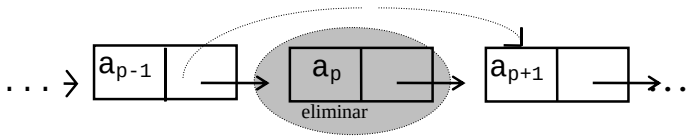
1- p = Primera(L) (cuando p=L)
2- p = Fin(L) (cuando p=nulo)

Para el caso general ($p \neq$ nulo y $p \neq L$) debemos considerar lo siguiente: Como debemos insertar el elemento x en la posición p, lo más conveniente es abrir una nueva celda a continuación de la que tiene el elemento a_p en la posición p, copiar a_p en la celda que se creó y por último ubicar a x en la celda apuntada por p. De la manera propuesta, la inserción en una posición que no sea Fin(L) se realizará en un tiempo de orden constante.

Para el caso p = Fin(L), es decir cuando la inserción de x se desea realizar a continuación del último elemento de L, no queda otra solución mejor que recorrer toda la lista para poder acceder a la última celda y enlazar en su campo siguiente una nueva célula que alojará a x. Este caso particular hace que el tiempo de ejecución de insertar sea del orden de n.

```
PROCEDIMIENTO Insertar ( x: TipoElemento;
                         p: Posicion;
                         var L: Lista)
aux: Posicion

Si p=L entonces (* p es primera posición *)
        aux := L;
        CrearCelda(L);
        Celda(L).elemento:= x;
        Celda(L).sig := aux

Sino Si p<>nulo  (* caso general *)
        aux := Celda(p).sig;
        CrearCelda(Celda(p).sig);
        Celda(Celda(p).sig).elemento := Celda(p).elemento;
        Celda(p).elemento := x;
        Celda(Celda(p).sig).sig := aux

Sino  (* p es Fin(L) *)
        aux := L;
        mientras Celda(aux).sig <> nulo hacer
          aux := Celda(aux).sig;
        CrearCelda(Celda(aux).sig);
        Celda(Celda(aux).sig).elemento := x;
        Celda(Celda(aux).sig).sig := nulo
FIN Insertar.
```

Para eliminar el elemento de la posición p, habrá que acceder a la célula anterior a la que contiene dicho elemento, pues se deberá modificar el campo que tiene el enlace a la siguiente célula.



Para poder acceder a la célula anterior a la que contiene el elemento a eliminar, se deberá invocar a la operación Anterior(p) o se deberá recorrer la lista hasta encontrar la mencionada célula. En cualquiera de los dos casos, el tiempo de ejecución resultará del orden de n.

```
PROCEDIMIENTO Eliminar ( p: Posicion;
                         var L: Lista)
var aux: Posicion:
Si L <> nulo entonces {la lista no es vacía}
        Si p = L entonces { primera posición}
                L := Celda(L).sig;
                EliminarCelda (p)
        Sino
                aux := L;
                mientras Celda(aux).sig ≠nulo y Celda(aux).sig ≠ p hacer
                aux := Celda(aux).sig;

                Si Celda(aux).sig = p entonces
                        Celda(aux).sig := Celda(Celda(aux).sig).sig;
                        EliminarCelda(p)

                Sino Error
FIN Eliminar
```

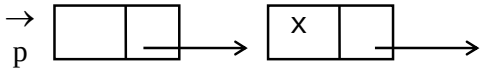
Para implementar la operación Localizar(x, L), observemos que se debe recorrer la lista hasta encontrar la primer ocurrencia de x, en caso de que x sea un elemento de la lista, o alcanzar la última célula de L en caso contrario. La naturaleza de la operación que se propone hace que al algoritmo que se obtenga no se le pueda exigir un tiempo de ejecución mejor que del orden de n. La confección del algoritmo queda como ejercicio.

Como podemos observar, parte de las expectativas planteadas para pensar en una estructura dinámica para representar una lista, no se ven satisfechas con esta representación. Nos referimos expresamente a la posibilidad de insertar o eliminar un elemento en cualquier parte de la lista en un tiempo constante. Del análisis del tiempo de ejecución de los algoritmos planteados para las correspondientes operaciones se desprende que en ambos casos es del orden de n.

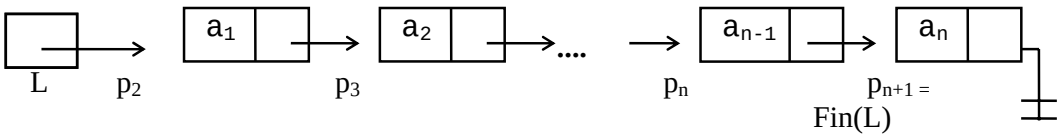
Para solucionar ese problema es conveniente definir el concepto de posición de manera diferente

3- Usando enlaces - Segunda versión

Se define posición p del elemento x como el enlace a la célula que en el campo siguiente tiene el enlace a la célula que contiene al elemento x.
Gráficamente:



Si bien esta definición de posición es menos natural que la presentada en el punto 2- , tiene ventajas sobre aquella. Antes de mencionar esas ventajas, analicemos las n+1 posiciones válidas en L= <a_1, a_2,...,a_n>

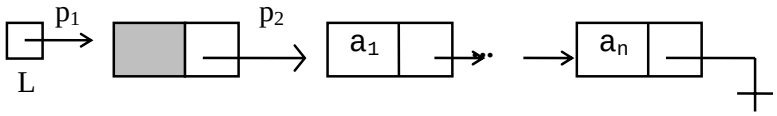


Si n>0, L contendrá la posición de a_2.

Como Fin(L) es la posición que sigue a la posición del elemento a_n, es natural que tenga el valor del enlace a la última célula.

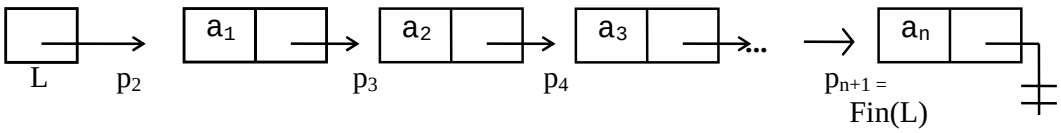
Observemos que quedan definidas así todas las posiciones menos la primera. Se puede optar entre dos soluciones diferentes:

a) Se considera por convención la primera posición nulo, cualquiera sea L.
b) Agregar al comienzo de la lista una célula que no contendrá ningún elemento (la llamaremos célula de encabezamiento). De esta manera aparecen en la estructura n+1 enlaces no nulos, que son los que se necesitan para las n+1 posiciones diferentes. De esta manera L contendrá el valor de la primera posición.



Desarrollaremos en este punto (3) la implementación del TDA Lista según la solución propuesta en a), y dejamos la propuesta en b) para desarrollarla en el punto 4.

Gráficamente:



posicion de a_1 = nulo

Observemos que en todos los casos deberemos tratar la primera posición como un caso particular.

Se presentan a continuación los procedimientos correspondientes a las operaciones Primera, Siguiente, Insertar y Eliminar. Los algoritmos de las operaciones restantes quedan propuestas como ejercicio.

```
PROCEDIMIENTO Primera (L: Lista) : Posicion
    Retornar (nulo)
FIN Primera

PROCEDIMIENTO Siguiente(p: Posicion;
                        L: Lista      ) : Posicion;
    Si p = nulo entonces Retornar(L)
    Sino  Retornar (Celda(p).sig)

FIN Siguiente

PROCEDIMIENTO Insertar( x: TipoElemento;
                        p: : Posicion;
                        Var L: Lista)
aux: Posicion;
    Si p = nulo entonces (*primera posición*)
            aux := L;
            CrearCelda (L);
            Celda(L).elemento := x;
            Celda(L).sig := aux

    Sino    aux := Celda(p).sig;
            CrearCelda (Celda(p).sig);
            Celda(Celda(p).sig).elemento := x;
            Celda(Celda(p).sig).sig := aux
FIN Insertar

PROCEDIMIENTO Eliminar (p: Posicion;
                        Var L: Lista);
aux: Posicion;
    Si L ≠ nulo entonces
            Si p = nulo entonces
                    aux := L;
                    L := Celda(L).Sig;

            Sino
                    aux := Celda(p).sig;
                    Celda(p).sig := Celda(Celda(p).sig).sig;

             EliminarCelda (aux)
FIN Eliminar
```

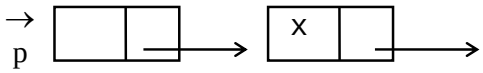
Se puede observar que las dos operaciones de actualización (Insertar y Eliminar), tienen en este caso tiempo de ejecución constante.

4- Usando enlaces - Tercera versión

Siguiendo el desarrollo de la idea planteada en el punto 3-b se define:

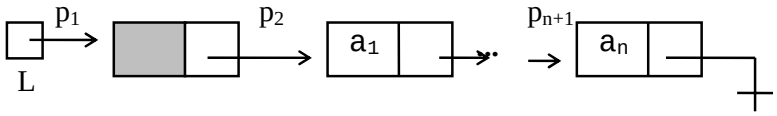
- Posición p del elemento x: el enlace a la célula que en el campo siguiente tiene el enlace a la célula que contiene al elemento x.

Gráficamente:



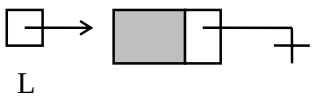
- Toda lista L contendrá una primer célula sin elemento, llamada célula de encabezamiento.

Gráficamente:



De esta manera L contendrá siempre la posición Primera(L) y el enlace que apunta a la célula que contiene en el campo sig el valor nulo, da el valor a Fin(L).

Una lista vacía estará constituída solamente por la célula de encabezamiento



L = < >
Primer(L) = Fin(L)

Observemos que en este caso, al considerar una célula de encabezamiento, no aparecen casos particulares en los algoritmos.

El desarrollo de los algoritmos correspondientes a las operaciones del TDA Lista para esta representación están desarrollados en [Aho 83] pág 45-47 para el caso particular de una implementación de los enlaces usando el tipo puntero.

Análisis del tiempo de ejecución de las operaciones:

El tiempo de ejecución de las operaciones Localizar(x,L) e Imprime(L) resulta en ambos casos del orden de n. Esto se debe a la naturaleza de las mismas, ya que ambas plantean necesariamente un recorrido de la lista.

El resto de las operaciones, con excepción de Anterior y Fin tienen tiempo de ejecución constante.

El tiempo de ejecución de las operaciones Anterior y Fin es del orden de n en ambos casos, ya que para obtener el correspondiente valor es necesario recorrer la lista desde el comienzo hasta alcanzarlo.

Una solución para que el tiempo de ejecución de Fin sea constante consiste en considerar el encabezamiento de la lista conteniendo además de un enlace a la primer célula, otro apuntando a la última.

Una solución para que el tiempo de ejecución de Anterior sea constante es que cada célula contenga además de un enlace a la siguiente, un enlace a la anterior.

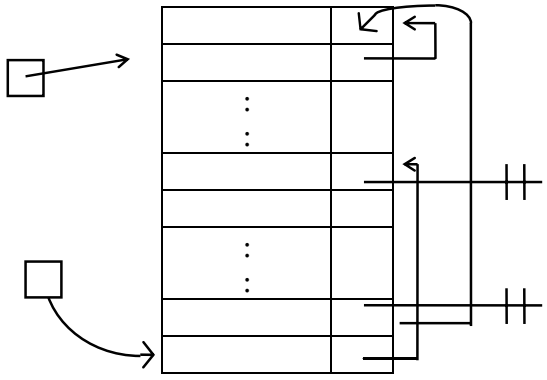
5- Implementación de los enlaces usando punteros

La implementación de las propuestas de los puntos 2, 3 y 4 con punteros es inmediata.
Si se hace en Modula-2 basta tener en cuenta las siguientes equivalencia:

Lenguaje de diseño	Modula-2
p: enlace a Celda	p:POINTER TO TipoCelda
celda(p)	p^
CrearCelda(p)	Allocate(p,SIZE(p^))
EliminarCelda(p)	Deallocate(p, SIZE(p^))
nulo	NIL

6- Implementación de los enlaces usando cursores

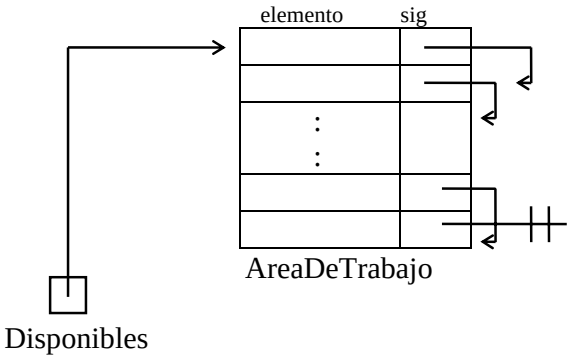
Cuando una implementación de listas usando arreglos no resulta adecuada (por ejemplo si se espera hacer un gran número de inserciones o eliminaciones al comienzo o en la mitad de la lista) y si el lenguaje de implementación no cuenta con una herramienta como los punteros, una solución es usar una estructura enlazada con cursores.
Para ello se define, como ya vimos, un arreglo que identificaremos en este caso como AreaDeTrabajo, y cuyas componentes alojarán las células de las listas del tipo Lista.
Gráficamente



AreaDeTrabajo

Para hacer un tratamiento similar al que se hace cuando se trabaja con enlaces, consideraremos:

- Inicialmente todas las células del AreaDeTrabajo están enlazadas en un encadenamiento de células que identificaremos por Disponibles



AreaDeTrabajo y Disponibles se implementarán con variables globales, evitando así que los parámetros de los procedimientos con los que se implementan las operaciones deban cambiar según la implementación que se considere.
Cada vez que sea necesario contar con una nueva celda para una lista, así como se invoca al procedimiento Allocate cuando se está trabajando con enlaces, se dispondrá en este caso del procedimiento CrearCelda(p), que implementaremos a continuación.

La ejecución de CrearCelda(p) dejará en p la dirección (índice de una componente de AreaDeTrabajo) de una celda y desconectará la mencionada celda de Disponibles.

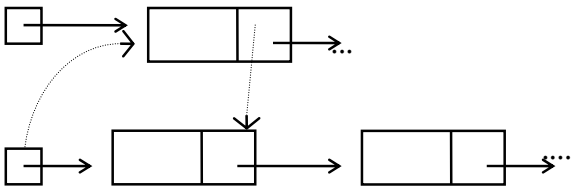
```
PROCEDIMIENTO CrearCelda ( var p: Posicion)
  Si Disponibles = 0 entonces
                    Error('AreaDeTrabajo no tiene mas celulas')
  Sino
      p := Disponibles;
      Disponibles := AreaDeTrabajo[Disponibles].sig
FIN CrearCelda
```

Cuando una célula es eliminada de una lista, deberá 'devolverse' al encadenamiento de células Disponibles para que pueda ser luego reusada.

De manera análoga a cuando se trabaja con punteros y se usa el procedimiento Deallocate, se usará en este caso el procedimiento EliminarCelda(p), cuyo efecto es encadenar en Disponibles la célula que se encuentra en la componente de índice p del AreaDeTrabajo.

Gráficamente:

p



Disponibles

```
PROCEDIMIENTO EliminarCelda ( p: Posicion);
| Si p ≠ 0 entonces
|              | AreaDeTrabajo[p].sig := Disponibles;
|              | Disponibles := p
|              |_____________________________________
|
FIN EliminarCelda
```

Teniendo en cuenta las consideraciones anteriores se realiza entonces:

- Definición de la estructura :

```
const  LongitudMaxima = ---- (* valor adecuado*)
       nulo = 0;

type   Posicion = Integer;

       TipoCelda = record
                     elemento: TipoElemento;
                     sig: Posicion
                   end;

       Lista = Posicion;

var    AreaDeTrabajo:  array[1.. LongitudMaxima] of TipoCelda;
       Disponibles: Lista
```

Se necesitará de la ejecución de un procedimiento de inicialización para encadenar las celdas y dejar a Disponibles apuntando a la primer celda del encadenamiento.

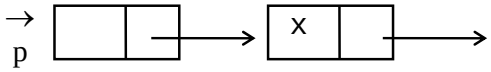
```
PROCEDIMIENTO Inicializar;
| PARA i := 1 hasta LongitudMaxima - 1 HACER
|   |AreaDeTrabajo[i].sig := i+1;
| AreaDeTrabajo[LongitudMaxima] := nulo;
| Disponibles := 1
FIN Inicializar
```

La ejecución de Inicializar deja a AreaDeTrabajo y a Disponibles en condiciones para el desarrollo de la aplicación.

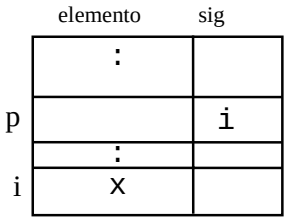
Podrían realizarse distintas implementaciones, de manera análoga a las presentadas al trabajar con enlaces, pero ya analizadas en esa oportunidad las ventajas de la implementación presentada en el punto 4- (con enlaces: tercera versión), desarrollaremos la equivalente enlazando en este caso, la estructura con cursores.

Podemos definir por lo tanto : posición p del elemento x como el índice de la componente del AreaDeTrabajo que en el campo sig tiene el índice de la componente que en el campo elemento aloja a x.

Gráficamente:



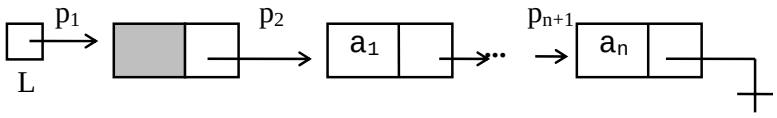
Observar que en este caso los enlaces por ser cursores son índices del arreglo AreaDeTrabajo y las células son componentes de dicho arreglo. La representación gráfica anterior es una abstracción de la siguiente:



AreaDeTrabajo

Con el propósito de poder hacer un tratamiento uniforme de todos los casos, se considerará entonces, una célula de encabezamiento (sin elemento) como primer célula de la lista.

Gráficamente:



Por lo tanto el valor de Fin(L) será el índice de la célula que en el campo sig tiene el cursor nulo, y L contendrá el valor de Primera(L). Observar que Primera(L) es la posición del primer elemento de la lista si ésta no es vacía y su valor coincide con el de Fin(L) en caso contrario.

- Implementación de las operaciones

```
PROCEDIMIENTO CrearListaVacia (var L: Lista);
|CrearCelda(L);
|AreaDeTrabajo[L].sig := nulo
FIN CrearListaVacia

PROCEDIMIENTO Primera(L: Lista) : Posicion;
|
```

```
  Si L = nulo entonces Error
  Sino Retornar(L)
FIN Primera

PROCEDIMIENTO Siguiente ( p: Posicion;
                          L: Lista) : Posicion;
Si L = nulo entonces Error
Sino
    Si AreaDeTrabajo[p].sig = nulo entonces Error
    Retornar(AreaDeTrabajo[p].sig)
FIN Siguiente

PROCEDIMIENTO Anterior( p: Posicion;
                        L: Lista) : Posicion;
var aux: Posicion;
Si (p = nulo) o (p = L) entonces Error
Sino
    aux := L;
    Mientras (AreaDeTrabajo[aux].sig ≠ nulo) y
             (AreaDeTrabajo[aux].sig ≠ p) hacer
    aux := AreaDeTrabajo[aux].sig
    Si AreaDeTrabajo[aux].sig ≠ nulo entonces Retornar(aux)
    Sino Error
FIN Anterior

PROCEDIMIENTO Localizar ( x: TipoElemento;
                          L: Lista ) : Posicion;
var aux: Posicion;
Si L ≠ nulo entonces
    aux := L;
    Mientras (AreaDeTrabajo[aux].sig ≠ nulo) y
             (AreaDeTrabajo [AreaDeTrabajo[aux].sig].elemento ≠ x)
    hacer
        aux := AreaDeTrabajo[aux].sig;

    Retornar(aux)
FIN Localizar
```

Observación:
El procedimiento Localizar es correcto si se lo implementa en Modula-2, donde la evaluación de una expresión booleana se hace siguiendo el camino más corto, es decir en este caso si la subexpresión AreaDeTrabajo[aux].sig ≠ nulo toma el valor falso y no se evalúa la segunda expresión (AreaDeTrabajo[AreaDeTrabajo[aux].sig].elemento≠x) para dar el valor falso correspondiente a la expresión booleana completa. Si la implementación se hiciese en lenguaje Pascal, el lazo debería plantearse de la siguiente manera:

```
   encontre := false;
   Mientras (AreaDeTrabajo[aux].sig <> nulo) y
            (not encontre) hacer
   ┌ Si AreaDeTrabajo[AreaDeTrabajo[aux].sig].elemento = x
   │ entonces   encontre := true
   └ Sino aux := AreaDeTrabajo[aux].sig;
   Localizar := aux

PROCEDIMIENTO Insertar ( x: TipoElemento;
                         p:  posicion;
                         L: Lista );
var aux : Posicion;
 aux := AreaDeTrabajo[p].sig;
 CrearCelda (AreaDeTrabajo[p].sig);
AreaDeTrabajo[AreaDeTrabajo[p].sig].elemento := x
AreaDeTrabajo[AreaDeTrabajo[p].sig].sig := aux

FIN Insertar

PROCEDIMIENTO Eliminar( p:  posicion;
                        L: Lista ) : Posicion;
var aux : Posicion;
 aux := AreaDeTrabajo[AreaDeTrabajo[p].sig].sig
 EliminarCelda (AreaDeTrabajo[p].sig);
AreaDeTrabajo[p].sig:= aux

FIN Eliminar
```

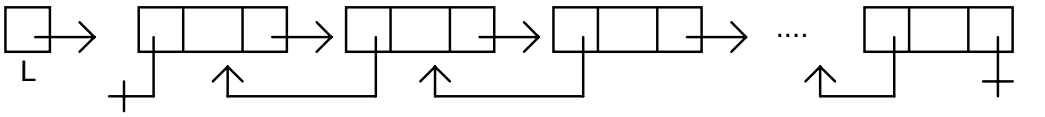
La implementación de las restantes operaciones queda propuesta como ejercicio.

7- Listas doblemente enlazadas

En el sección 4 se presentó un análisis del tiempo de ejecución de las operaciones para esa implementación de listas con enlaces. Se observó allí, que para dos de las operaciones (Anterior y Fin) los algoritmos que se podían lograr con la estructura propuesta tenían tiempo de ejecución del orden de n.

Para el caso de la operación Anterior, esto se puede mejorar considerando cada célula con dos enlaces; uno apuntando a la célula siguiente y otro a la célula anterior. De esta forma con un costo adicional de espacio (un apuntador por cada celda) y un costo extra de tiempo (constante) para las operaciones de actualización, podrá ejecutarse la operación Anterior en tiempo constante. Esto permitirá recorrer una lista en ambos sentidos de manera eficiente. La estructura que se obtiene se la conoce con el nombre de lista doblemente enlazada.

Representación gráfica de una lista doblemente enlazada

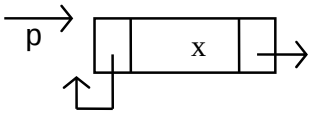


Definición del concepto de posición:

Como cada célula (excepto la primera) tiene un apuntador a la célula anterior, podría considerarse en principio, sin que perjudicase el tiempo de ejecución de las operaciones de actualización, la definición de posición más natural, es decir:

Posición del elemento x = apuntador a la célula que contiene a x.

Gráficamente:

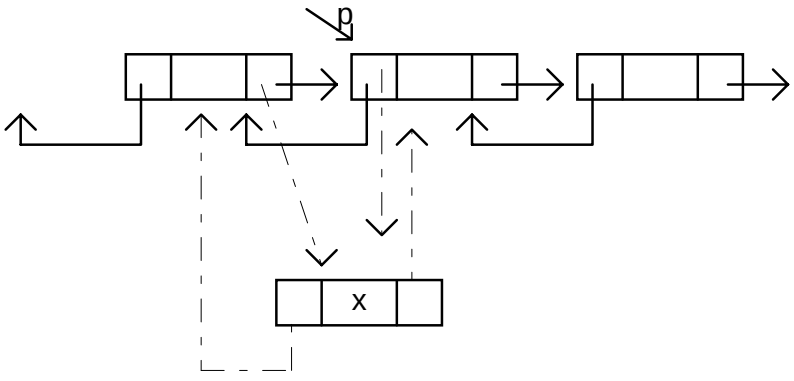


Se puede definir entonces la estructura de la siguiente manera:

```
Posicion = enlace a TipoCelda;
Lista = Posicion;
TipoCelda =  registro con
                elemento: TipoElemento;
                siguiente,
                anterior: Posicion
```

Para realizar actualizaciones de la lista, en posiciones que no sean las extremas, deberá realizarse lo siguiente:

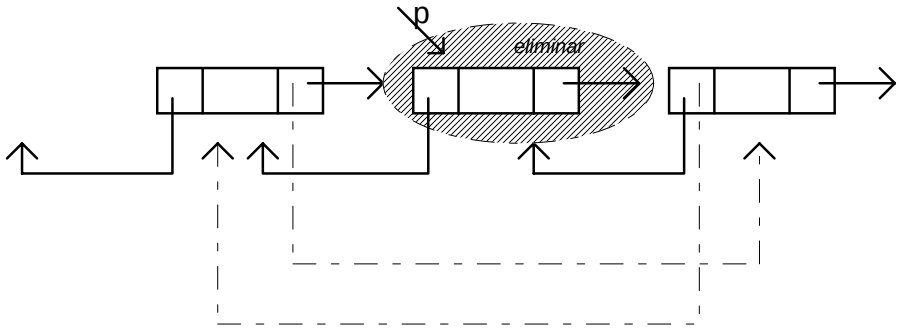
- Para insertar el elemento x en la posición p de la lista L:




```
CrearCelda (Celda(Celda(p).anterior).siguiente);
Celda(Celda(Celda(p).anterior).siguiente).elemento := x;
Celda(Celda(Celda(p).anterior).siguiente).siguiente :=p;
```

Celda(Celda(Celda(p).anterior).siguiente).anterior := Celda(p).anterior;
Celda(p).anterior := Celda(Celda(p).anterior).siguiente

- Para eliminar el elemento de la posición p de la lista L:

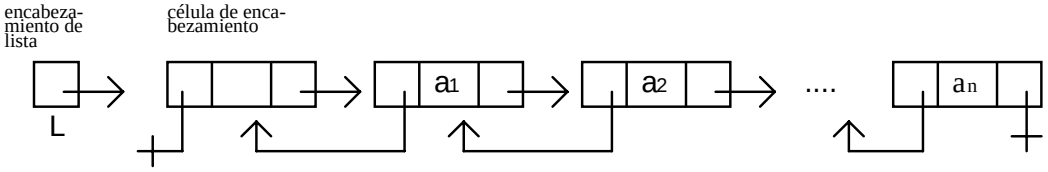


Celda(Celda(p).anterior).siguiente := Celda(p).siguiente;
Celda(Celda(p).siguiente).anterior := Celda(p).anterior; (#)
EliminarCelda(p)

Observaciones:

- La inserción o eliminación en la primera posición deberá tratarse como caso especial ya que en vez de modificarse el valor de un campo identificado por Celda(Celda(p).anterior).siguiente , se cambia el valor de la variable L. Para evitar que las actualizaciones en la primera posición deban ser tratadas especialmente, es conveniente considerar una primer célula de encabezamiento (sin elemento).

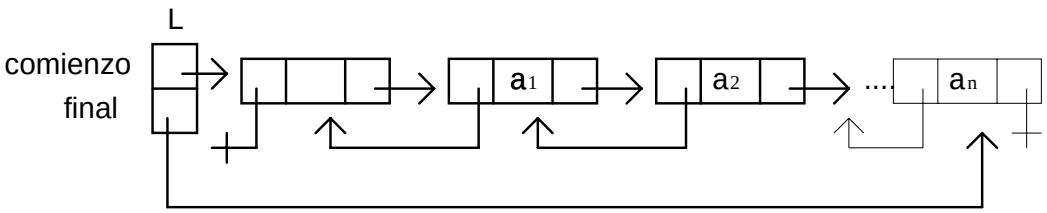
- La eliminación del último elemento no requiere de la asignación (#), ya que no hay una célula siguiendo a la última. En ese caso particular, la ejecución de dicha asignación produciría un error ya que la variable apuntada Celda(Celda(p).sig) no existe allí.



- El valor de Fin(L) es nulo, cualquiera se la lista L, por lo tanto la operación Insertar un elemento en la posición Fin(L) resultará con un tiempo de ejecución del orden de n, a menos que se adopte alguna política especial, tal como la de considerar un enlace extra apuntando a la última célula. Dicho enlace podrá ubicarse tanto en el encabezamiento de la lista como en el campo anterior de la célula de encabezamiento.

7.1 Implementación con célula de encabezamiento y encabezamiento de lista con dos enlaces.

Si se considera el encabezamiento de la lista L compuesto de dos enlaces, uno apuntando a la primer célula de la lista y otro a la última es posible acceder a la última célula en tiempo constante. Si además se considera una célula de encabezamiento es posible además, tratar las actualizaciones que se realicen de manera uniforme, cualquiera sea la posición. Solamente al eliminar el último elemento hay que tener presente que no existe otra célula a continuación que deba ser actualizada y al insertar un elemento al final de la lista, el acceso a la última celda se hace mediante L.final.



La estructura se define entonces, de la siguiente manera:

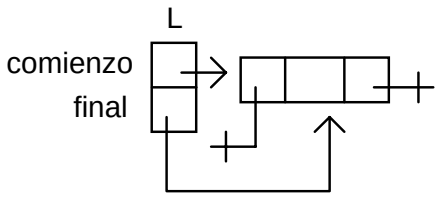
```
Posicion = enlace a TipoCelda;
Lista = registro con
            comienzo,
            final : Posicion;

TipoCelda =  registro con
                elemento: TipoElemento;
                siguiente,
                anterior: Posicion
```

En este caso, si bien el valor de Fin(L) es nulo, la inserción de un elemento en esa posición se podrá hacer en tiempo constante ya que se deberá enlazar una nueva celda al campo Celda(L.final).siguiente.

El valor de Primera(L) en esta representación está dado por el enlace que está en el campo siguiente de la célula de encabezamiento (la apuntada por L).

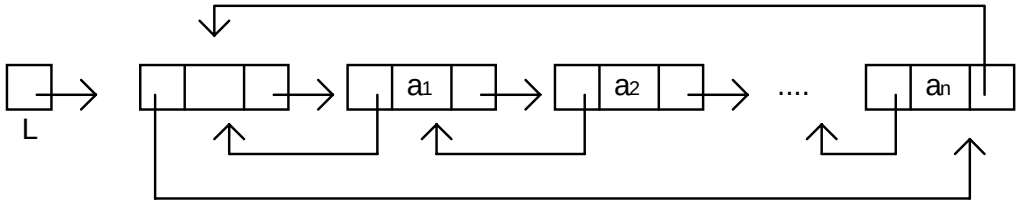
Hay que tener presente que cualquier actualización de la lista deberá dejar los enlaces de L, comienzo y final, apuntando a la primer y última célula de la lista respectivamente.

Una lista vacía constará solamente del encabezamiento con los dos enlaces y una única célula sin elemento (célula de encabezamiento).



7.2 Estructura circular con célula de encabezamiento

En esta representación el campo anterior de la célula de encabezamiento tiene un enlace a la última célula, y en el campo siguiente de la última célula hay un enlace a la primera. De esta manera se obtiene una estructura que si bien por sus enlaces es circular, de acuerdo a la definición de las posiciones especiales Primera y Fin, se comportará como las presentadas anteriormente.

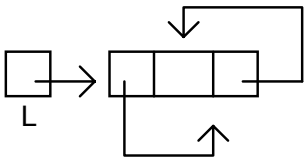


Como se puede observar, la definición de la estructura dada inicialmente se mantiene para esta representación, como así también el concepto de posición que coincide con el de la representación anterior

En este caso, el valor de Fin(L) está dado por el enlace que se encuentra en el campo siguiente de la última célula (observar que coincide con el valor de L).

El valor de Primera(L) es también en esta representación el valor del enlace que está en el campo siguiente de la célula de encabezamiento. (Celda(L).siguiente).

Una lista vacía constará solamente del encabezamiento de la lista y de una célula de encabezamiento con enlaces a sí misma en los campos anterior y siguiente.



La implementación de las operaciones del TDA: Lista correspondientes a las estructuras definidas en 6.1 y 6.2 quedan propuestas como ejercicio.

Implementación usando cursores

Para hacerse la implementación usando cursores deberá tenerse en cuenta la siguiente definición:

```
const  LongitudMaxima = ---- (* valor adecuado *)
       nulo = 0;

type   Posicion = Integer;

       TipoCelda = record
                     elemento: TipoElemento;
                     anterior,
                     siguiente: Posicion
                   end;
       Lista = posicion;

var    AreaDeTrabajo:  array[1.. LongitudMaxima] of TipoCelda;
       Disponibles: Lista
```

Los procedimientos Inicializar, CrearCelda y EliminarCelda presentados en el punto 6 son válidos también para este caso. Observar que no es necesario mantener a Disponibles doblemente enlazada, ya que sólo se desea mantener las celdas que están libres encadenadas de tal forma de poder obtener la primera de todas ellas o ingresar una nueva al comienzo del encadenamiento. El enlace queda hecho por el campo siguiente.

Se propone como ejercicio la implementación del TDA: Lista con cursores y usando las estructuras propuestas en 7.1 y 7.2

Bibliografía

[Aho83] Aho,Alfred; Hopcroft,John and Ullman Jeffrey. Estructuras de datos y algoritmos. Addison-Wesley.

[Señ95] Perla Señas y Alejandro García; “Estructuras Enlazadas: Una aproximación independiente de la implementación”, 3er Ateneo de Profesores Universitarios de Computación y Sistemas. Universidad Nacional del Sur. 1995.