UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

TDA Lista Doblemente Enlazada

Objetivo: Comprender el uso e implementación del TDA Lista doblemente Enlazada (LDE).

Introducción

Para iniciar tengamos en mente lo visto en Lista Simplemente Enlazada (LSE) respecto a los usos de una lista de manera tal de no volver inventar la rueda. Habiendo hecho hincapié en ello, marquemos la principales diferencias entre estos dos TDA:

- A diferencia de la LSE, una LDE puede ser recorrida en ambos sentidos, es decir, la LDE se puede recorrer de adelante hacia atrás y viceversa.
- En la LDE, el puntero a la lista apunta al último nodo insertado. Recordemos en una LSE, el puntero a la lista siempre apunta al primer elemento ingresado en la lista.

Implementación del TDA Lista Doblemente Enlazada

Para permitir recorrer la lista en ambos sentidos, cada nodo de la lista debe poseer dos punteros, uno al siguiente nodo de la lista, y uno al nodo anterior. En la figura 1 se muestra la implementación de un nodo de la lista. No olvidemos que cada nodo posee además de los punteros al los nodos anterior y siguiente, un puntero de tipo **`void`** a la estructura de datos que contiene y una variable que indica el tamaño en bytes del bloque de información.

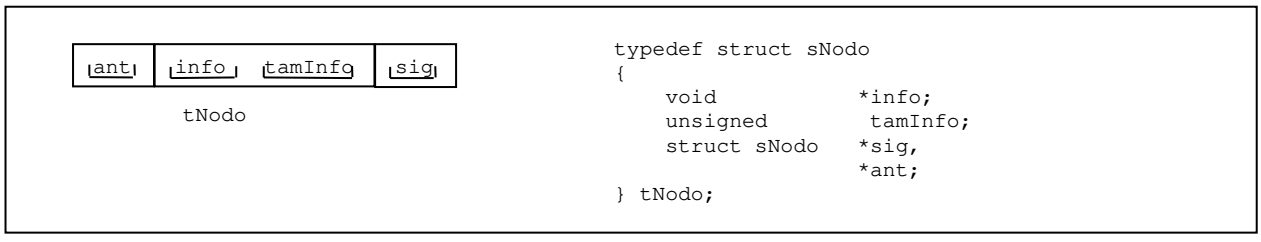

Figura 1. Implementación del nodo de la lista doblemente enlazada.

Cuando ya se ha creado una lista y se han insertado nodos en ella, podremos determinar cuáles son los nodos inicial y final de dicha lista a través de los punteros al nodo anterior y al nodo siguiente. Es decir, si nos encontramos con un nodo cuyo puntero al nodo anterior es **`null`**, nos encontramos al inicio de la lista. En el caso de encontrarnos con un nodo cuyo puntero a nodo siguiente es **`null`**, estamos en el nodo final de la lista. Es importante remarcar esto, porque en una LDE, el puntero a la lista puede estar apuntando a cualquier nodo de la lista. En la figura 2 se muestra un ejemplo con una lista con cuatro nodos insertados en ella.

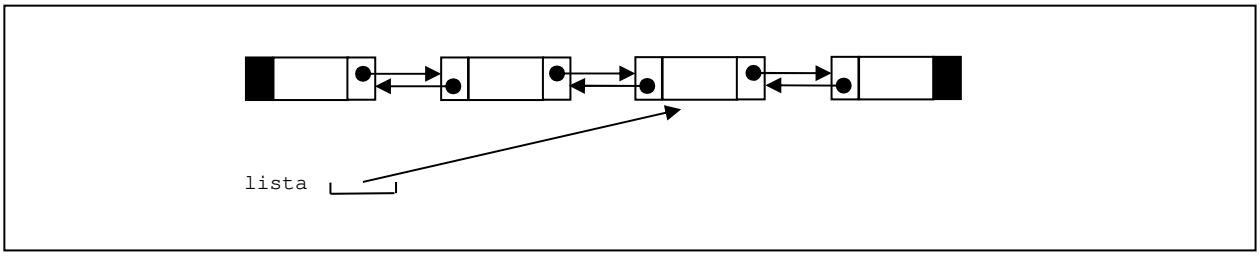

*Figura 2. Lista doblemente enlazada con cuatro elementos insertados. Los punteros a nodo anterior y siguiente rellenos en color negro simbolizan que contienen **`NULL`***

Ejemplo de aplicación: Para ejemplificar el uso e implementación de algunas primitivas de LDE, supondremos que debemos cargar sobre la lista una serie de productos que fueron modelados bajo el tipo de dato **`tProducto`**. La figura 3 muestra la definición del tipo de dato y sus funciones asociadas.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct
{
    long id_producto;
    char descripcion[30];
    float precioUnitario;
    int cantidad;

}tProducto;

void crearProducto(tProducto *, long, char*, float, int);
void mostrarProducto(const void *);
int comparaProductoPorID(const void *, const void *);
int acumularProducto(void **, unsigned *, const void *, unsigned);
```

Figura 3. Definición del tipo de dato TProducto y sus funciones asociadas.

En la figura 4 se pueden observar las diferentes primitivas a implementar. Recordemos que no son las únicas primitivas que se pueden implementar sobre una LDE, ya que este tipo de TDA permite cualquier primitiva que el negocio requiera.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIN_MEM     0
#define TODO_BIEN   1
#define CLA_DUP     2

typedef struct sNodo
{
    void            *info;
    unsigned         tamInfo;
    struct sNodo    *sig,
                    *ant;
} tNodo;

typedef tNodo *tLista;

void crearLista(tLista *p);
int  vaciarLista(tLista *p);
int  listaVacia(const tLista *p);
int  listaLlena(const tLista *p, unsigned cantBytes);
int  insertarAlFinal(tLista *p, const void *d, unsigned cantBytes);
int  insertarAlComienzo(tLista *p, const void *d, unsigned cantBytes);
int  mostrarDeIzqADer(const tLista *p,
                      void(*mostrar)(const void *));
int  mostrarDeDerAIzq(const tLista *p,
                      void(*mostrar)(const void *));
int  insertarEnOrden(tLista *p, const void *d, unsigned cantBytes,
                     int (*comparar)(const void *, const void *),
                     int (*acumular)(void **, unsigned *,
                                     const void *, unsigned));
void ordenarLista(tLista *p, int (*comparar)(const void *, const void *));
int eliminarPorClave(tLista *p, void *d, unsigned cantBytes,
                     int (*comparar)(const void *, const void *));
```

Figura 4. Primitivas de LDE a implementar.

La figura 5 muestra las llamadas a las primitivas de la lista doblemente enlazada. Les proponemos que observen las salidas por pantalla para complementar la comprensión. En el **Anexo I** se encuentran los listados completos del programa.

Ejercicio propuesto: Modifique el programa del ejemplo anterior de manera tal que la lista de productos provenga de un archivo binario. Los registros de productos contenidos en el archivo no se encuentran ordenados.

Ejercicio propuesto: Se debe diseñar un sistema que permita actualizar un archivo maestro de productos a través de un archivo de movimientos. El archivo maestro de productos no se encuentra ordenado, y para evitar la búsqueda secuencial por el código de producto, se desea mejorar el tiempo de acceso a éste archivo. Para ello, se debe implementar un mecanismo de indexación sobre una lista doblemente enlazada. El diagrama de bloques del sistema se encuentra expuesto en la figura 5.

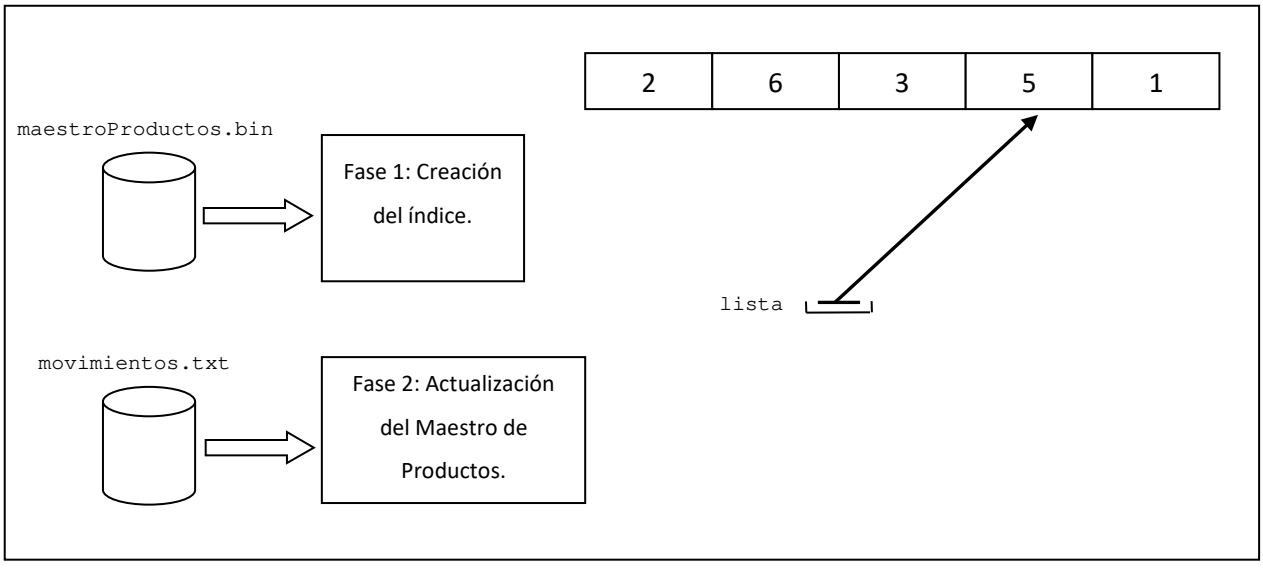

Figura 5. Diagrama de bloque del sistema de actualización para el ejercicio propuesto.

En la Fase 1 se deberá crear el índice en la lista. Para ello, la lista deberá contener el código del producto y el número de registro que el producto ocupa en el disco. La lista deberá estar ordenada por el código de producto. El tipo de dato contenido en la lista será **`tIndice`**.

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

Durante la Fase 2 se realizará la actualización del maestro de productos. La actualización consistirá en modificar la cantidad de producto existente en el archivo maestro. La cantidad deberá actualizarse a partir de lo indicado en el archivo de texto **`movimientos.txt`**. Este archivo posee los campos de sus registros separados por el caracter **`"|"`** (pipe), y el contenido de sus campos es:

`COD_PRODUCTO|CANTIDAD|TIPO_OPERACION`

En donde **`TIPO_OPERACION`** puede ser "V" (Venta, se debe restar la cantidad) o "R" (Reposición, se debe sumar la cantidad). Los movimientos no poseen ningún orden útil y pueden estar repetidos.

Cada vez que se lea una línea (que corresponde a un registro) del archivo de movimientos, debe realizarse el parseo de sus campos, luego de determina la posición del producto en el archivo maestro de productos a través de la lista y se realiza la actualización conforme al tipo de operación señalada. El proceso debe continuar hasta que se alcance el final del archivo de movimientos.

Debe imprimir en pantalla el contenido del archivo maestro de productos al inicio del proceso y posteriormente al final en dónde deben verse reflejados los cambios.

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

Anexo I: Listado del programa de ejemplo de implementación de primitivas

main.h

```
1  #ifndef MAIN_H_INCLUDED
2  #define MAIN_H_INCLUDED
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #include "listaDoble.h"
8  #include "tipoProducto.h"
9
10 #endif // MAIN_H_INCLUDED
```

main.c

```
1  #include "main.h"
2
3  int main()
4  {
5      tProducto p7, p1, p2, p3, p4, p5, pe;
6      tLista lista;
7
8      /** Crear la lista */
9      crearLista(&lista);
10
11     /** Verificar si la lista esta vacia */
12     if(listaVacia(&lista))
13     {
14         printf("\nLa lista se encuentra vacia.\n\n");
15     }
16
17     /** Verificar si la lista esta llena */
18     if(listaLlena(&lista, sizeof(tProducto)))
19     {
20         printf("\nLa lista se encuentra llena.\n\n");
21     }
22
23     /** Insertar registros en la lista */
24     crearProducto(&p1, 1, "Producto 1", 111.11, 11);
25     insertarAlComienzo(&lista, &p1, sizeof(tProducto));
26     crearProducto(&p5, 5, "Producto 5", 555.5, 40);
27     insertarAlComienzo(&lista, &p5, sizeof(tProducto));
28     crearProducto(&p2, 2, "Producto 2", 22.22, 22);
29     insertarAlComienzo(&lista, &p2, sizeof(tProducto));
30
31     crearProducto(&p7, 7, "Producto 7", 0.701, 74);
32     insertarAlFinal(&lista, &p7, sizeof(tProducto));
33
```

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

```
34      /** Mostrar la lista */
35      printf("\n\n#### Mostrar lista de Derecha a Izquierda\n\n");
36      mostrarDeDerAIzq(&lista, mostrarProducto);
37      printf("\n\n#### Mostrar lista de Izquierda a Derecha\n\n");
38      mostrarDeIzqADer(&lista, mostrarProducto);
39
40      /** Ordenar la lista */
41      printf("\n\n#### Lista ordenada:\n\n");
42      ordenarLista(&lista, comparaProductoPorID);
43      mostrarDeIzqADer(&lista, mostrarProducto);
44
45      /** Insertar en orden */
46      crearProducto(&p4, 4, "Producto 4", 44.0, 70);
47      insertarEnOrden(&lista, &p4, sizeof(tProducto),
48                      comparaProductoPorID, acumularProducto);
49      crearProducto(&p3, 3, "Producto 3", 33.2, 550);
50      insertarEnOrden(&lista, &p3, sizeof(tProducto),
51                      comparaProductoPorID, acumularProducto);
52      crearProducto(&p4, 4, "Producto 4", 44.0, 30);
53      insertarEnOrden(&lista, &p4, sizeof(tProducto),
54                      comparaProductoPorID, acumularProducto);
55
56      printf("\n\n#### Lista luego de la insercion en orden\n\n");
57      mostrarDeIzqADer(&lista, mostrarProducto);
58
59      /** Eliminar producto */
60      pe.id_producto = 2;
61      eliminarPorClave(&lista, &pe, sizeof(tProducto), comparaProductoPorID);
62      printf("\n\n#### Datos del producto eliminado de la lista\n\n");
63      mostrarProducto(NULL);
64      mostrarProducto(&pe);
65
66      printf("\n\n#### Lista luego de la eliminacion del producto\n\n");
67      mostrarDeIzqADer(&lista, mostrarProducto);
68
69      /** Vaciar la lista */
70      vaciarLista(&lista);
71
72      printf("\nFin\n");
73      return 0;
74  }
```

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

tipoProducto.h

```
1  #ifndef TIPOPRODUCTO_H_INCLUDED
2  #define TIPOPRODUCTO_H_INCLUDED
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7
8  typedef struct
9  {
10     long id_producto;
11     char descripcion[30];
12     float precioUnitario;
13     int cantidad;
14
15 }tProducto;
16
17 void crearProducto(tProducto *, long, char*, float, int);
18 void mostrarProducto(const void *);
19 int comparaProductoPorID(const void *, const void *);
20 int acumularProducto(void **, unsigned *, const void *, unsigned);
21
22 #endif // TIPOPRODUCTO_H_INCLUDED
```

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

tipoProducto.c

```c
#include "tipoProducto.h"

void crearProducto(tProducto *d,
                   long id_producto,
                   char *descripcion,
                   float precioUnitario,
                   int cantidad)
{
    d->id_producto = id_producto;
    strcpy(d->descripcion, descripcion);
    d->precioUnitario = precioUnitario;
    d->cantidad = cantidad;
}

void mostrarProducto(const void *dv)
{
    tProducto *d = (tProducto *) dv;
    if(d == NULL)
    {
      printf("ID_PRODUCTO DESCRIPCION                    PRECIO_UNITARIO CANTIDAD\n");
      printf("----------- ------------------------------ --------------- --------\n");
    }
    else
    {
        printf("%011ld %-*.*s %06.2f          %03d\n", d->id_producto,
                                                       sizeof(d->descripcion)-1,
                                                       sizeof(d->descripcion)-1,
                                                       d->descripcion,
                                                       d->precioUnitario,
                                                       d->cantidad);
    }
}

int comparaProductoPorID(const void *p1v, const void *p2v)
{
    tProducto *p1 = (tProducto *) p1v;
    tProducto *p2 = (tProducto *) p2v;
    return p1->id_producto - p2->id_producto;
}

int acumularProducto(void **p1v, unsigned *tp1, const void *p2v, unsigned tp2)
{
    tProducto **p1 = (tProducto **) p1v;
    tProducto *p2 = (tProducto *) p2v;
    (*p1)->cantidad += p2->cantidad;
    return 1;
}

```

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

listaDoble.h

```
 4  #ifndef LISTADOBLE_H_
 5  #define LISTADOBLE_H_
 6
 7  #include <stdio.h>
 8  #include <stdlib.h>
 9  #include <string.h>
10
11  #define SIN_MEM     0
12  #define TODO_BIEN   1
13  #define CLA_DUP     2
14
15  typedef struct sNodo
16  {
17      void           *info;
18      unsigned        tamInfo;
19      struct sNodo   *sig,
20                     *ant;
21  } tNodo;
22
23  typedef tNodo *tLista;
24
25  void crearLista(tLista *p);
26  int  vaciarLista(tLista *p);
27  int  listaVacia(const tLista *p);
28  int  listaLlena(const tLista *p, unsigned cantBytes);
29  int  insertarAlFinal(tLista *p, const void *d, unsigned cantBytes);
30  int  insertarAlComienzo(tLista *p, const void *d, unsigned cantBytes);
31  int  mostrarDeIzqADer(const tLista *p,
32                        void(*mostrar)(const void *));
33  int  mostrarDeDerAIzq(const tLista *p,
34                        void(*mostrar)(const void *));
35  int  insertarEnOrden(tLista *p, const void *d, unsigned cantBytes,
36                       int (*comparar)(const void *, const void *),
37                       int (*acumular)(void **, unsigned *,
38                                       const void *, unsigned));
39  void ordenarLista(tLista *p, int (*comparar)(const void *, const void *));
40  int eliminarPorClave(tLista *p, void *d, unsigned cantBytes,
41                       int (*comparar)(const void *, const void *));
42
43  #endif // LISTADOBLE_H_
```

listaDoble.c

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

```c
/* ---------------------------------o---x---o---------------------------------
 *          listaDoble.c     LISTA DOBLEMENTE ENLAZADA
 * ---------------------------------o---x---o--------------------------------- */

#include "listaDoble.h"

void crearLista(tLista *p)
{
    *p = NULL;
}


int  vaciarLista(tLista *p)
{
    int     cant = 0;
    tNodo  *act = *p;

    if(act)
    {
        while(act->ant)
            act = act->ant;
        while(act)
        {
            tNodo *aux = act->sig;

            free(act->info);
            free(act);
            act = aux;
            cant++;
        }
        *p = NULL;
    }
    return cant;
}


int  listaVacia(const tLista *p)
{
    return *p == NULL;
}


int  listaLlena(const tLista *p, unsigned cantBytes)
{
     tNodo  *nue = (tNodo *)malloc(sizeof(tNodo));
     void   *aux = malloc(cantBytes);

     free(nue);
     free(aux);
     return aux == NULL || nue == NULL;
}

```

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

```
53
54  int  insertarAlFinal(tLista *p, const void *d, unsigned cantBytes)
55  {
56      tNodo  *act = *p,
57             *nue;
58
59      if(act)
60          while(act->sig)
61              act = act->sig;
62      if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
63         (nue->info = malloc(cantBytes)) == NULL)
64      {
65          free(nue);
66          return 0;
67      }
68      memcpy(nue->info, d, cantBytes);
69      nue->tamInfo = cantBytes;
70      nue->sig = NULL;
71      nue->ant = act;
72      if(act)
73          act->sig = nue;
74      *p = nue;
75      return 1;
76  }
77
78
79  int  insertarAlComienzo(tLista *p, const void *d, unsigned cantBytes)
80  {
81      tNodo  *act = *p,
82             *nue;
83
84      if(act)
85          while(act->ant)
86              act = act->ant;
87      if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
88         (nue->info = malloc(cantBytes)) == NULL)
89      {
90          free(nue);
91          return 0;
92      }
93      memcpy(nue->info, d, cantBytes);
94      nue->tamInfo = cantBytes;
95      nue->sig = act;
96      nue->ant = NULL;
97      if(act)
98          act->ant = nue;
99      *p = nue;
100     return 1;
101 }
102
```

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

```
103
104 int  mostrarDeIzqADer(const tLista *p,
105                       void(*mostrar)(const void *))
106 {
107     tNodo  *act = *p;
108     int     cant = 0;
109
110     if(act)
111     {
112         mostrar(NULL);
113         while(act->ant)
114             act = act->ant;
115         while(act)
116         {
117             mostrar(act->info);
118             act = act->sig;
119             cant++;
120         }
121     }
122     return cant;
123 }
124
125
126 int  mostrarDeDerAIzq(const tLista *p,
127                       void(*mostrar)(const void *))
128 {
129     tNodo  *act = *p;
130     int     cant = 0;
131
132     if(act)
133     {
134         mostrar(NULL);
135         while(act->sig)
136             act = act->sig;
137         while(act)
138         {
139             mostrar(act->info);
140             act = act->ant;
141             cant++;
142         }
143     }
144     return cant;
145 }
146
147
```

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

```
148 int  insertarEnOrden(tLista *p, const void *d, unsigned cantBytes,
149                      int (*comparar)(const void *, const void *),
150                      int (*acumular)(void **, unsigned *,
151                                      const void *, unsigned))
152 {
153     tNodo  *act = *p,
154            *sig,
155            *ant,
156            *nue;
157
158     if(act == NULL)
159     {
160         ant = NULL;
161         sig = NULL;
162     }
163     else
164     {
165         int aux;
166         while(act->sig && comparar(act->info, d) < 0)
167             act = act->sig;
168         while(act->ant && comparar(act->info, d) > 0)
169             act = act->ant;
170         aux = comparar(act->info, d);
171         if(aux == 0)
172         {
173             *p = act;
174             if(acumular)
175                 if(acumular(&act->info, &act->tamInfo, d, cantBytes) == 0)
176                     return SIN_MEM;
177             return CLA_DUP;
178         }
179         if(aux < 0)
180         {
181             ant = act;
182             sig = act->sig;
183         }
184         else
185         {
186             ant = act->ant;
187             sig = act;
188         }
189     }
190     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
191        (nue->info = malloc(cantBytes)) == NULL)
192     {
193         free(nue);
194         return SIN_MEM;
195     }
196     memcpy(nue->info, d, cantBytes);
197     nue->tamInfo = cantBytes;
198     nue->sig = sig;
```

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

```
199     nue->ant = ant;
200     if(sig)
201         sig->ant = nue;
202     if(ant)
203         ant->sig = nue;
204     *p = nue;
205     return TODO_BIEN;
206 }
207
208
209 void ordenarLista(tLista *p, int (*comparar)(const void *, const void *))
210 {
211     tNodo  *act = *p,
212            *sup = NULL,
213            *inf = NULL;
214     int     marca = 1;
215
216     if(act == NULL)
217         return;
218     while(act->ant)
219         act = act->ant;
220     while(marca)
221     {
222         marca = 0;
223         while(act->sig != sup)
224         {
225             if(comparar(act->info, act->sig->info) > 0)
226             {
227                 void       *inf = act->info;
228                 unsigned    tam = act->tamInfo;
229
230                 marca = 1;
231                 act->info = act->sig->info;
232                 act->sig->info = inf;
233                 act->tamInfo = act->sig->tamInfo;
234                 act->sig->tamInfo = tam;
235             }
236             act = act->sig;
237         }
238         sup = act;
```

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

```
239         while(act->ant != inf)
240         {
241             if(comparar(act->info, act->ant->info) < 0)
242             {
243                 void       *inf = act->info;
244                 unsigned    tam = act->tamInfo;
245
246                 marca = 1;
247                 act->info = act->ant->info;
248                 act->ant->info = inf;
249                 act->tamInfo = act->ant->tamInfo;
250                 act->ant->tamInfo = tam;
251             }
252             act = act->ant;
253         }
254         inf = act;
255     }
256 }
257
258
259 int eliminarPorClave(tLista *p, void *d, unsigned cantBytes,
260                      int (*comparar)(const void *, const void *))
261 {
262     tNodo  *act = *p;
263     int     aux;
264
265     if(act == NULL)
266     {
267         return 0;       /** Lista vacia */
268     }
269
270     while(act->sig && comparar(act->info, d) < 0)
271         act = act->sig;
272     while(act->ant && comparar(act->info, d) > 0)
273         act = act->ant;
274
275     aux = comparar(act->info, d);
276
277     if(aux == 0)        /** Encontro la clave */
278     {
279         tNodo  *ant = act->ant,
280                *sig = act->sig;
281
282         if(ant)
283             ant->sig = sig;
284         if(sig)
285         {
286             sig->ant = ant;
287             *p = sig;
288         }
289         else
290             *p = ant;
```

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

```
291
292             memcpy(d, act->info, cantBytes);
293             free(act->info);
294             free(act);
295             return 1;
296         }
297         return 0;
298     }
```

Índice de figuras

Bibliografía

[1] B. W. Kernighan y D. M. Ritchie, El lenguaje de programación C, Pearson Educación, 1991.

[2] H. M. Deitel y P. J. Deitel, Cómo programar en C/C+, Pearson Educación, 1995.

[3] Herbert Schildt, Turbo C/C++ 3.1 Manual de referencia, Osborne/McGraw-Hill, 1994.