```c
/* --------------------o---x---o--------------------
 *          main.h
 * --------------------o---x---o-------------------- */
#ifndef MAIN_H_
#define MAIN_H_

#include <stdio.h>

#include "productos/productos.h"
#include "texto/texto.h"
#include "lista/lista.h"


void probarIngresarYMostrarProd(void);

void probarPonerAdelanteVerUltimoVerPrimeroYVaciar(void);

void probarPonerAtrasSacarUltimoSacarPrimeroYMostrar(void);

void probarPonerEnOrdenMostrarAlRevesYOrdenar(void);


void probarIngresarYMostrarTexto(void);

void probarPonerEnOrdenYMostrarTexto(void);

void probarPonerAlFinalYElimDuplNoConsecTexto(void);

void probarPonerAlFinalYElimDuplSiConsecTexto(void);

void probarPonerAlFinalYElimTodosDuplNoConsecTexto();

void  probarPonerAlFinalYElimTodosDuplSiConsecTexto();


#endif
/* --------------------o---x---o--------------------
 *          main.c
 * --------------------o---x---o-------------------- */

#include "main.h"


int main(void)
{/**
    probarIngresarYMostrarProd();

    probarPonerAdelanteVerUltimoVerPrimeroYVaciar();

    probarPonerAtrasSacarUltimoSacarPrimeroYMostrar();

    probarPonerEnOrdenMostrarAlRevesYOrdenar();

    probarIngresarYMostrarTexto();

    probarPonerEnOrdenYMostrarTexto();

    probarPonerAlFinalYElimDuplNoConsecTexto();

    probarPonerAlFinalYElimDuplSiConsecTexto();
*/
    probarPonerAlFinalYElimTodosDuplNoConsecTexto();

    probarPonerAlFinalYElimTodosDuplSiConsecTexto();

    return 0;
}


```

```c
void probarIngresarYMostrarProd(void)
{
    tProd   prod;
    int     result,
            cant = 0;

    puts("\nProbando ingresar productos y mostrar productos");
    result = ingresarProducto(&prod);
    if(result)
        mostrarProducto(NULL);
    while(result)
    {
        mostrarProducto(&prod);
        result = ingresarProducto(&prod);
        cant++;
    }
    fprintf(stdout, "Se mostraron %d productos.\n", cant);
}


void probarPonerAdelanteVerUltimoVerPrimeroYVaciar(void)
{
    tLista      lista;
    tProd       prod;
    int         cant = 0;

    puts("\nFUNCION: probarPonerAdelanteVerUltimoVerPrimeroYVaciar");
    crearLista(&lista);

    printf("Insertando al comienzo de la lista\n");
    while(ingresarProducto(&prod) &&
          ponerAlComienzo(&lista, &prod, sizeof(tProd)))
    {
        if(cant == 0)
            mostrarProducto(NULL);
        mostrarProducto(&prod);
        cant++;
    }
    printf("Se insertaron %d productos en la lista\n", cant);
    if(verUltimoLista(&lista, &prod, sizeof(prod)))
    {
        printf("Ultimo de la lista\n");
        mostrarProducto(NULL);
        mostrarProducto(&prod);
    }
    if(verPrimeroLista(&lista, &prod, sizeof(prod)))
    {
        printf("Primero de la lista\n");
        mostrarProducto(NULL);
        mostrarProducto(&prod);
    }
    vaciarLista(&lista);
}


void probarPonerAtrasSacarUltimoSacarPrimeroYMostrar(void)
{
    tLista      lista;
    tProd       prod;
    int         cant = 0;

    puts("\nFUNCION: probarPonerAtrasSacarUltimoSacarPrimeroYMostrar");
    crearLista(&lista);
    printf("Insertando al final de la lista\n");
    while(ingresarProducto(&prod) &&
          ponerAlFinal(&lista, &prod, sizeof(tProd)))
    {
        if(cant == 0)
            mostrarProducto(NULL);
```

```
139         mostrarProducto(&prod);
140         cant++;
141     }
142     printf("Se insertaron %d productos en la lista\n", cant);
143     if(sacarPrimeroLista(&lista, &prod, sizeof(prod)))
144     {
145         printf("Primero de la lista\n");
146         mostrarProducto(NULL);
147         mostrarProducto(&prod);
148     }
149     if(sacarUltimoLista(&lista, &prod, sizeof(prod)))
150     {
151         printf("Ultimo de la lista\n");
152         mostrarProducto(NULL);
153         mostrarProducto(&prod);
154     }
155     cant = mostrarLista(&lista, mostrarProducto);
156     if(cant)
157         printf("Se mostraron %d productos\n", cant);
158     vaciarLista(&lista);
159 }
160
161
162 void probarPonerEnOrdenMostrarAlRevesYOrdenar(void)
163 {
164     tLista      lista;
165     tProd       prod;
166     int         cant = 0,
167                 dupl = 0;
168
169     puts("\nFUNCION: probarPonerEnOrdenMostrarAlRevesYOrdenar");
170     crearLista(&lista);
171
172     printf("Insertando en orden por clave del producto\n");
173     while(ingresarProducto(&prod))
174
175     {
176         int result = ponerEnOrdenOAcumular(&lista, &prod, sizeof(tProd),
177                                            compararProdXCodProd, NULL);
178         if(cant == 0)
179             mostrarProducto(NULL);
180         mostrarProducto(&prod);
181         if(result == TODO_BIEN)
182             cant++;
183         if(result == CLA_DUP)
184             dupl++;
185     }
186     printf("Se insertaron %d productos en la lista y hubo %d duplicados\n",
187            cant, dupl);
188     cant = mostrarListaAlReves(&lista, mostrarProducto);
189     if(cant)
190         printf("Se mostraron %d productos\n", cant);
191     puts("Ordenando por descripcion del producto");
192     ordenarLista(&lista, compararProdXDescrip);
193     cant = mostrarListaAlReves(&lista, mostrarProducto);
194     if(cant)
195         printf("Se mostraron %d productos\n", cant);
196     vaciarLista(&lista);
197 }
198
199
200 /** se supone que las líneas de texto son de un máximo de 299 carácteres
201  **     y no tienen marca de fin de registro (\n).
202  **/
203 void probarIngresarYMostrarTexto(void)
204 {
205     char    linea[300];
206     int     cant = 0;
207
```

```
208       puts("\nProbando probarIngresarYMostrarTexto");
209       while(ingresarTexto(linea))
210       {
211    ///    puts(linea);
212           mostrarTexto(linea);
213           cant++;
214       }
215       fprintf(stdout, "Se mostraron %d lineas de texto.\n", cant);
216   }
217
218   void probarPonerEnOrdenYMostrarTexto(void)
219   {
220       tLista      lista;
221       char        texto[300];
222       int         cant = 0,
223                   dupl = 0;
224
225       puts("\nFUNCION: probarPonerEnOrdenYMostrarTexto");
226       crearLista(&lista);
227
228       printf("Insertando en orden por primer palabra\n");
229       while(ingresarTexto(texto))
230       {
231           int result = ponerEnOrdenOAcumular(&lista, texto,
232                                              strlen(texto) + 1,
233                                              compararXPriPalTexto,
234                                              acumularConcatenandoLineasDeTexto);
235           mostrarTexto(texto);
236           if(result == TODO_BIEN)
237               cant++;
238           if(result == CLA_DUP)
239               dupl++;
240       }
241       printf("Se insertaron %d lineas de texto y hubo %d duplicados\n",
242              cant, dupl);
243       cant = mostrarLista(&lista, mostrarTexto);
244       if(cant)
245           printf("Se mostraron %d lineas de texto\n", cant);
246       vaciarLista(&lista);
247   }
248
249   void probarPonerAlFinalYElimDuplNoConsecTexto(void)
250   {
251       tLista      lista;
252       char        texto[300];
253       int         cant = 0;
254
255       puts("\nFUNCION: probarPonerAlFinalYElimDuplNoConsecTexto");
256       crearLista(&lista);
257
258       printf("Insertando al final\n");
259       while(ingresarTexto(texto))
260           if(ponerAlFinal(&lista, texto, strlen(texto) + 1))
261               cant++;
262       printf("Se insertaron %d lineas de texto\n", cant);
263       cant = mostrarLista(&lista, mostrarTexto);
264       if(cant)
265           printf("Se mostraron %d lineas de texto\n", cant);
266       cant = eliminarDupliYAcumNoConsec(&lista, compararXPriPalTexto,
267                                         acumularConcatenandoLineasDeTexto);
268   /// cant = eliminarDupliYAcumNoConsec(&lista, compararXPriPalTexto, NULL);
269       if(cant)
270           printf("Se eliminaron %d nodos acumulando\n", cant);
271       cant = mostrarLista(&lista, mostrarTexto);
272       if(cant)
273           printf("Quedaron %d nodos de texto\n", cant);
274       vaciarLista(&lista);
275   }
276
```

```
277  void probarPonerAlFinalYElimDuplSiConsecTexto(void)
278  {
279      tLista      lista;
280      char        texto[300];
281      int         cant = 0;
282
283      puts("\nFUNCION: probarPonerAlFinalYElimDuplSiConsecTexto");
284      crearLista(&lista);
285
286      printf("Insertando al final\n");
287      while(ingresarTexto(texto))
288          if(ponerAlFinal(&lista, texto, strlen(texto) + 1))
289              cant++;
290      printf("Se insertaron %d lineas de texto\n", cant);
291      cant = mostrarLista(&lista, mostrarTexto);
292      if(cant)
293          printf("Se mostraron %d lineas de texto\n", cant);
294      cant = eliminarDupliYAcumSiConsec(&lista, compararXPriPalTexto,
295                                        acumularConcatenandoLineasDeTexto);
296  /// cant = eliminarDupliYAcumSiConsec(&lista, compararXPriPalTexto, NULL);
297      if(cant)
298          printf("Se eliminaron %d nodos acumulando\n", cant);
299      cant = mostrarLista(&lista, mostrarTexto);
300      if(cant)
301          printf("Quedaron %d nodos de texto\n", cant);
302      vaciarLista(&lista);
303  }
304
305  void probarPonerAlFinalYElimTodosDuplNoConsecTexto(void)
306  {
307      tLista      lista;
308      char        texto[300];
309      int         cant = 0;
310
311      puts("\nFUNCION: probarPonerAlFinalYElimTodosDuplNoConsecTexto");
312      crearLista(&lista);
313
314      printf("Insertando al final\n");
315      while(ingresarTexto(texto))
316          if(ponerAlFinal(&lista, texto, strlen(texto) + 1))
317              cant++;
318      printf("Se insertaron %d lineas de texto\n", cant);
319      cant = mostrarLista(&lista, mostrarTexto);
320      if(cant)
321          printf("Se mostraron %d lineas de texto\n", cant);
322      cant = eliminarTodosDupliNoConsec(&lista, compararXPriPalTexto);
323      if(cant)
324          printf("Se eliminaron %d nodos\n", cant);
325      cant = mostrarLista(&lista, mostrarTexto);
326      if(cant)
327          printf("Quedaron %d nodos de texto\n", cant);
328      vaciarLista(&lista);
329  }
330
331  void probarPonerAlFinalYElimTodosDuplSiConsecTexto(void)
332  {
333      tLista      lista;
334      char        texto[300];
335      int         cant = 0;
336
337      puts("\nFUNCION: probarPonerAlFinalYElimTodosDuplSiConsecTexto");
338      crearLista(&lista);
339
340      printf("Insertando al final\n");
341      while(ingresarTexto(texto))
342          if(ponerAlFinal(&lista, texto, strlen(texto) + 1))
343              cant++;
344      printf("Se insertaron %d lineas de texto\n", cant);
345      cant = mostrarLista(&lista, mostrarTexto);
```

```
    if(cant)
        printf("Se mostraron %d lineas de texto\n", cant);
    cant = eliminarTodosDupliSiConsec(&lista, compararXPriPalTexto);
    if(cant)
        printf("Se eliminaron %d nodos\n", cant);
    cant = mostrarLista(&lista, mostrarTexto);
    if(cant)
        printf("Quedaron %d nodos de texto\n", cant);
    vaciarLista(&lista);
}
/* --------------------o---x---o--------------------
 *          productos.h
 * --------------------o---x---o-------------------- */
#ifndef PRODUCTOS_H_
#define PRODUCTOS_H_

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


typedef struct
{
    char    codProd[11],
            descrip[46];
} tProd;

int ingresarProducto(tProd *d);

void mostrarProducto(const void *d);

int  compararProdXCodProd(const void *d1, const void *d2);

int  compararProdXDescrip(const void *d1, const void *d2);


#endif
/* --------------------o---x---o--------------------
 *          productos.c
 * --------------------o---x---o-------------------- */

#include "productos.h"


int ingresarProducto(tProd *d)
{
    static const tProd productos[] = {
        ///1234567890    123456789 123456789 123456789 123456789 12345
        { "clavoro3/4", "Clavo de oro 24 kilates de 3/4 de pulgada" },
        { "martillo3K", "Martillo bolita con saca clavos de 3 kilos"},
        { "yesoalam1",  "Alambre de yeso de un milimetro de espesor" },
        { "vidrem-15",  "Remache de vidrio de 1,5 milimetros" },
        { "plom-telgo", "Plomada de poliestireno expandido" },
        { "limagoma17", "Lima de goma de 17 pulgadas"} };
    static int posi = 0;

    if(posi == sizeof(productos) / sizeof(tProd))
    {
        posi = 0;
        return 0;
    }
    *d = productos[posi];
    posi++;

    return 1;
}

void mostrarProducto(const void *d)
{
    tProd  *dProd = (tProd *)d;
```

```
415         if(d)
416             fprintf(stdout,
417                     "%-*s %-*s ...\n",
418                     sizeof(dProd->codProd) - 1, dProd->codProd,
419                     sizeof(dProd->descrip) - 1, dProd->descrip);
420         else
421             fprintf(stdout,
422                     "%-*.*s %-*.*s ...\n",
423                     sizeof(dProd->codProd) - 1, sizeof(dProd->codProd) - 1,
424                         "Cod. Producto",
425                     sizeof(dProd->descrip) - 1, sizeof(dProd->descrip) - 1,
426                         "Descripcion del producto");
427     }
428
429     int compararProdXCodProd(const void *d1, const void *d2)
430     {
431         tProd  *p = (tProd *)d1,
432                *q = (tProd *)d2;
433
434         return strcmpi(p->codProd, q->codProd);
435     }
436
437     int compararProdXDescrip(const void *d1, const void *d2)
438     {
439     /** tProd  *p = (tProd *)d1,
440                *q = (tProd *)d2;
441
442         return strcmpi(p->descrip, q->descrip);*/
443         return strcmpi(((tProd *)d1)->descrip, ((tProd *)d2)->descrip);
444     }
445
446     /* --------------------o---x---o--------------------
447      *          texto.h
448      * --------------------o---x---o-------------------- */
449     #ifndef TEXTO_H_
450     #define TEXTO_H_
451
452     #include <string.h>
453     #include <stdio.h>
454     #include <ctype.h>
455     #include <stdlib.h>
456
457     int  ingresarTexto(char *d);
458
459     void mostrarTexto(const void *d);
460
461     int  compararXPriPalTexto(const void *d1, const void *d2);
462
463     void acumularConcatenandoLineasDeTexto(void **dest, const void *orig,
464                                            unsigned *tamDest, unsigned tamOrig);
465
466
467
468     #endif
469     /* --------------------o---x---o--------------------
470      *          texto.c
471      * --------------------o---x---o-------------------- */
472
473      #include "texto.h"
474
475
476     int ingresarTexto(char *d)
477     {
478         char *texto[] = { "Himno Nacional Argentino",
479                           "",
480                           "Oid mortales",
481                           "El grito sagrado",
482                           "Libertad",
483                           "Libertad",
```

```
                        "Libertad",
                        "Oid el ruido de rotas cadenas",
                        "Ved en trono a la noble igualdad",
                        "Ya a su trono dignisimo abrieron",
                        "Las provincias unidas del Sud",
                        "Y los libres del mundo responden",
                        "Al gran pueblo argentino salud",
                        "Y los libres del mundo responden",
                        "Al gran pueblo argentino salud",
                        "",
                        "Sean eternos los laureles",
                        "Que supimos conseguir",
                        "Sean eternos los laureles",
                        "Que supimos conseguir",
                        "Coronados de gloria vivamos",
                        "O juremos con gloria morir",
                        "O juremos con gloria morir",
                        "O juremos con gloria morir",
                        "O juremos con gloria morir",
                        "",
                        "",
                        "Letra: Vicente Lopez y Planes",
                        "Musica: Blas Parera",
                        NULL };
    static int posi = 0;

    if(texto[posi] == NULL)
    {
        posi = 0;
        return 0;
    }
    strcpy(d, texto[posi]);
    posi++;

    return 1;
}

void mostrarTexto(const void *d)
{
    puts((const char *)d);
}

int compararXPriPalTexto(const void *d1, const void *d2)
{
    const char *p = (const char *)d1,
               *q = (const char *)d2;

    while(toupper(*p) == toupper(*q) && *p && isalpha(*p))
    {
        p++;
        q++;
    }
    return *p - *q;
}

void acumularConcatenandoLineasDeTexto(void **dest, const void *orig,
                                       unsigned *tamDest, unsigned tamOrig)
{
    const char *d1 = *(const char **)dest,
               *d2 = (const char *)orig;
    char       *res;

    res = (char *)malloc(*tamDest + tamOrig - 1);
    if(res == NULL)
        return;   /// ERROR catastófico de falta de memoria, no se concatena
    strcpy(res, d1);
    strcat(res, d2);
    free(*dest);
    *dest = (void *)res;
```

```
    *tamDest += tamOrig - 1;
}

/* ---------------------o---x---o---------------------
 *          lista.h
 * ---------------------o---x---o--------------------- */
#ifndef LISTA_H_
#define LISTA_H_

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

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

#define minimo( X , Y )     ( ( X ) <= ( Y ) ? ( X ) : ( Y ) )

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


void crearLista(tLista *p);

int  listaVacia(const tLista *p);

int  listaLlena(const tLista *p, unsigned cantBytes);

void vaciarLista(tLista *p);

int  ponerAlComienzo(tLista *p, const void *d, unsigned cantBytes);


int  sacarPrimeroLista(tLista *p, void *d, unsigned cantBytes);

int  verPrimeroLista(const tLista *p, void *d, unsigned cantBytes);

int  ponerAlFinal(tLista *p, const void *d, unsigned cantBytes);

int  sacarUltimoLista(tLista *p, void *d, unsigned cantBytes);

int  verUltimoLista(const tLista *p, void *d, unsigned cantBytes);


int  mostrarLista(const tLista *p, void (*mostrar)(const void *));

int  mostrarListaAlReves(const tLista *p, void (*mostrar)(const void *));

void ordenarLista(tLista *p, int (*comparar)(const void *, const void *));

int  ponerEnOrdenOAcumular(tLista *p, const void *d, unsigned cantBytes,
                           int (*comparar)(const void *, const void *),
                           void (*acumular)(void **, const void *,
                                            unsigned *, unsigned));

int  eliminarDupliYAcumNoConsec(tLista *p,
                                int (*comparar)(const void *, const void *),
                                void (*acumular)(void **, const void *,
                                                 unsigned *, unsigned));

int  eliminarDupliYAcumSiConsec(tLista *p,
                                int (*comparar)(const void *, const void *),
                                void (*acumular)(void **, const void *,
                                                 unsigned *, unsigned));
```

```
622
623   int  eliminarTodosDupliNoConsec(tLista *p,
624                                   int (*comparar)(const void *, const void *));
625
626   int  eliminarTodosDupliSiConsec(tLista *p,
627                                   int (*comparar)(const void *, const void *));
628
629
630
631   #endif
632   /* --------------------o---x---o---------------------
633    *          lista.c
634    * --------------------o---x---o--------------------- */
635
636   #include "lista.h"
637
638   void crearLista(tLista *p)
639   {
640       *p = NULL;
641   }
642
643   int  listaVacia(const tLista *p)
644   {
645       return *p == NULL;
646   }
647
648   int  listaLlena(const tLista *p, unsigned cantBytes)
649   {
650       tNodo  *aux = (tNodo *)malloc(sizeof(tNodo));
651       void   *info = malloc(cantBytes);
652
653       free(aux);
654       free(info);
655       return aux == NULL || info == NULL;
656   }
657
658   void vaciarLista(tLista *p)
659   {
660       while(*p)
661       {
662           tNodo *aux = *p;
663
664           *p = aux->sig;
665           free(aux->info);
666           free(aux);
667       }
668   }
669
670   int  ponerAlComienzo(tLista *p, const void *d, unsigned cantBytes)
671   {
672       tNodo *nue;
673
674       if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
675          (nue->info = malloc(cantBytes)) == NULL)
676       {
677           free(nue);
678           return 0;
679       }
680       memcpy(nue->info, d, cantBytes);
681       nue->tamInfo = cantBytes;
682       nue->sig = *p;
683       *p = nue;
684       return 1;
685   }
686
687   int  sacarPrimeroLista(tLista *p, void *d, unsigned cantBytes)
688   {
689       tNodo *aux = *p;
690
```

```
691         if(aux == NULL)
692             return 0;
693         *p = aux->sig;
694         memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
695         free(aux->info);
696         free(aux);
697         return 1;
698     }
699
700     int  verPrimeroLista(const tLista *p, void *d, unsigned cantBytes)
701     {
702         if(*p == NULL)
703             return 0;
704         memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
705         return 1;
706     }
707
708     int  ponerAlFinal(tLista *p, const void *d, unsigned cantBytes)
709     {
710         tNodo  *nue;
711
712         while(*p)
713             p = &(*p)->sig;
714         if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
715            (nue->info = malloc(cantBytes)) == NULL)
716         {
717             free(nue);
718             return 0;
719         }
720         memcpy(nue->info, d, cantBytes);
721         nue->tamInfo = cantBytes;
722         nue->sig = NULL;
723         *p = nue;
724         return 1;
725     }
726
727     int  sacarUltimoLista(tLista *p, void *d, unsigned cantBytes)
728     {
729         if(*p == NULL)
730             return 0;
731         while((*p)->sig)
732             p = &(*p)->sig;
733         memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
734         free((*p)->info);
735         free(*p);
736         *p = NULL;
737         return 1;
738     }
739
740     int  verUltimoLista(const tLista *p, void *d, unsigned cantBytes)
741     {
742         if(*p == NULL)
743             return 0;
744         while((*p)->sig)
745             p = &(*p)->sig;
746         memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
747         return 1;
748     }
749
750     int mostrarLista(const tLista *p, void (*mostrar)(const void *))
751     {
752         int     cant = 0;
753
754         if(*p)
755             mostrar(NULL);
756         while(*p)
757         {
758             mostrar((*p)->info);
759             p = &(*p)->sig;
```

```
760             cant++;
761         }
762         return cant;
763     }
764
765     int  mostrarListaAlReves(const tLista *p, void (*mostrar)(const void *))
766     {
767         if(*p)
768         {
769             int cant = mostrarListaAlReves(&(*p)->sig, mostrar);
770             mostrar((*p)->info);
771             return cant + 1;
772         }
773         mostrar(NULL);
774         return 0;
775     }
776
777     void ordenarLista(tLista *p, int (*comparar)(const void *, const void *))
778     {
779         int marca = 1;
780
781         if(*p == NULL)
782             return;
783         while(marca)
784         {
785             tLista *q = p;
786             marca = 0;
787             while((*q)->sig)
788             {
789                 if(comparar((*q)->info, (*q)->sig->info) > 0)
790                 {
791                     void       *infoAux = (*q)->info;
792                     unsigned    tamAux = (*q)->tamInfo;
793                     (*q)->info = (*q)->sig->info;
794                     (*q)->sig->info = infoAux;
795                     (*q)->tamInfo = (*q)->sig->tamInfo;
796                     (*q)->sig->tamInfo = tamAux;
797                     marca = 1;
798                 }
799                 q = &(*q)->sig;
800             }
801         }
802     }
803
804     int  ponerEnOrdenOAcumular(tLista *p, const void *d, unsigned cantBytes,
805                                int (*comparar)(const void *, const void *),
806                                void (*acumular)(void **, const void *,
807                                                 unsigned *, unsigned))
808     {
809         tNodo  *nue;
810
811         while(*p && comparar(d, (*p)->info) > 0)
812             p = &(*p)->sig;
813         if(*p && comparar(d, (*p)->info) == 0)
814         {
815             if(acumular)
816                 acumular(&(*p)->info, d, &(*p)->tamInfo, cantBytes);
817             return CLA_DUP;
818         }
819         if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
820            (nue->info = malloc(cantBytes)) == NULL)
821         {
822             free(nue);
823             return SIN_MEM;
824         }
825         memcpy(nue->info, d, cantBytes);
826         nue->tamInfo = cantBytes;
827         nue->sig = *p;
828         *p = nue;
```

```
829         return TODO_BIEN;
830     }
831
832     int  eliminarDupliYAcumNoConsec(tLista *p,
833                                     int (*comparar)(const void *, const void *),
834                                     void (*acumular)(void **, const void *,
835                                                      unsigned *, unsigned))
836     {
837         int cant = 0;
838
839         while(*p)
840         {
841             tLista *q = &(*p)->sig;
842             while(*q)
843             {
844                 if(comparar((*p)->info, (*q)->info) == 0)
845                 {
846                     tNodo  *aux = *q;
847
848                     if(acumular)
849                         acumular(&(*p)->info, aux->info,
850                                  &(*p)->tamInfo, aux->tamInfo);
851                     *q = aux->sig;
852                     free(aux->info);
853                     free(aux);
854                     cant++;
855                 }
856                 else
857                     q = &(*q)->sig;
858             }
859             p = &(*p)->sig;
860         }
861         return cant;
862     }
863
864     int  eliminarDupliYAcumSiConsec(tLista *p,
865                                     int (*comparar)(const void *, const void *),
866                                     void (*acumular)(void **, const void *,
867                                                      unsigned *, unsigned))
868     {
869         int cant = 0;
870
871         if(*p)
872             while((*p)->sig)
873             {
874                 if(comparar((*p)->info, (*p)->sig->info) == 0)
875                 {
876                     tNodo  *aux = (*p)->sig;
877
878                     if(acumular)
879                         acumular(&(*p)->info, aux->info,
880                                  &(*p)->tamInfo, aux->tamInfo);
881                     (*p)->sig = aux->sig;
882                     free(aux->info);
883                     free(aux);
884                     cant++;
885                 }
886                 else
887                     p = &(*p)->sig;
888             }
889         return cant;
890     }
891
892     int  eliminarTodosDupliNoConsec(tLista *p,
893                                     int (*comparar)(const void *, const void *))
894     {
895         int     cant = 0;
896         tNodo  *aux;
897
```

```
898         while(*p)
899         {
900             int elim = 0;
901             tLista *q = &(*p)->sig;
902             while(*q)
903             {
904                 if(comparar((*p)->info, (*q)->info) == 0)
905                 {
906                     aux = *q;
907                     *q = aux->sig;
908                     free(aux->info);
909                     free(aux);
910                     elim++;
911                 }
912                 else
913                     q = &(*q)->sig;
914             }
915             if(elim)
916             {
917                 aux = *p;
918                 *p = aux->sig;
919                 free(aux->info);
920                 free(aux);
921                 cant += elim + 1;
922             }
923             else
924                 p = &(*p)->sig;
925         }
926         return cant;
927     }
928
929     int  eliminarTodosDupliSiConsec(tLista *p,
930                                     int (*comparar)(const void *, const void *))
931     {
932         int     cant = 0,
933                 elim;
934         tNodo  *aux;
935
936         if(*p)
937             while((*p)->sig)
938             {
939                 elim = 0;
940                 while(comparar((*p)->info, (*p)->sig->info) == 0)
941                 {
942                     aux = (*p)->sig;
943                     (*p)->sig = aux->sig;
944                     free(aux->info);
945                     free(aux);
946                     elim++;
947                 }
948                 if(elim)
949                 {
950                     aux = *p;
951                     *p = aux->sig;
952                     free(aux->info);
953                     free(aux);
954                     cant += elim + 1;
955                 }
956                 else
957                     p = &(*p)->sig;
958             }
959         return cant;
960     }
961
962
963
```