```c
/* ---------------------------------o---x---o---------------------------------
 *          lista.h     declaración y primitivas del TDA LISTA
 *                      implementada en lista dinámica simplemente enlazada
 * ---------------------------------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


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);


#endif // LISTA_H_


/* ---------------------------------o---x---o--------------------------------- */

/* ---------------------------------o---x---o---------------------------------
 *          lista.c     definición y primitivas del TDA LISTA
 *                      implementada en lista dinámica simplemente enlazada
 * ---------------------------------o---x---o--------------------------------- */

#include "lista.h"


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


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


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


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

    free(aux);
    free(info);
    return aux == NULL || info == NULL;
```

```
85  }
86
87
88  void vaciarLista(tLista *p)
89  {
90      while(*p)
91      {
92          tNodo *aux = *p;
93
94          *p = aux->sig;
95          free(aux->info);
96          free(aux);
97      }
98  }
99
100
101 int  ponerAlComienzo(tLista *p, const void *d, unsigned cantBytes)
102 {
103     tNodo *nue;
104
105     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
106        (nue->info = malloc(cantBytes)) == NULL)
107     {
108         free(nue);
109         return 0;
110     }
111     memcpy(nue->info, d, cantBytes);
112     nue->tamInfo = cantBytes;
113     nue->sig = *p;
114     *p = nue;
115     return 1;
116 }
117
118
119 int  sacarPrimeroLista(tLista *p, void *d, unsigned cantBytes)
120 {
121     tNodo *aux = *p;
122
123     if(aux == NULL)
124         return 0;
125     *p = aux->sig;
126     memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
127     free(aux->info);
128     free(aux);
129     return 1;
130 }
131
132
133 int  verPrimeroLista(const tLista *p, void *d, unsigned cantBytes)
134 {
135     if(*p == NULL)
136         return 0;
137     memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
138     return 1;
139 }
140
141
142 int  ponerAlFinal(tLista *p, const void *d, unsigned cantBytes)
143 {
144     tNodo  *nue;
145
146     while(*p)
147         p = &(*p)->sig;
148     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
149        (nue->info = malloc(cantBytes)) == NULL)
150     {
151         free(nue);
152         return 0;
153     }
154     memcpy(nue->info, d, cantBytes);
155     nue->tamInfo = cantBytes;
156     nue->sig = NULL;
157     *p = nue;
158     return 1;
159 }
160
161
162 int  sacarUltimoLista(tLista *p, void *d, unsigned cantBytes)
163 {
164     if(*p == NULL)
165         return 0;
166     while((*p)->sig)
167         p = &(*p)->sig;
168     memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
```

```
169         free((*p)->info);
170         free(*p);
171         *p = NULL;
172         return 1;
173     }
174
175
176     int  verUltimoLista(const tLista *p, void *d, unsigned cantBytes)
177     {
178         if(*p == NULL)
179             return 0;
180         while((*p)->sig)
181             p = &(*p)->sig;
182         memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
183         return 1;
184     }
185
186
187     /* ------------------------------o---x---o------------------------------ */
188
189     /* ------------------------------o---x---o------------------------------
190      *          main.h     prueba del TDA LISTA con asignación dinámica de memoria
191      * ------------------------------o---x---o------------------------------ */
192
193     #ifndef MAIN_H_
194     #define MAIN_H_
195
196     #include <stdio.h>
197
198
199     #include "../../productos/productos.h"
200     #include "../../lineasDeTexto/lineasTexto.h"
201     #include "../../lista/lista.h"
202
203
204     void probarIngresarYMostrarProd(void);
205
206     void probarIngresarYMostrarTexto(void);
207
208     void probarPonerYSacarDeLista(void);
209
210
211     #endif // MAIN_H_
212
213
214     /* ------------------------------o---x---o------------------------------ */
215
216     /* ------------------------------o---x---o------------------------------
217      *          main.c     prueba del TDA LISTA con asignación dinámica de memoria
218      * ------------------------------o---x---o------------------------------ */
219
220     #include "main.h"
221
222
223     int main(void)
224     {
225         probarIngresarYMostrarProd();
226
227         probarIngresarYMostrarTexto();
228
229         probarPonerYSacarDeLista();
230
231         return 0;
232     }
233
234
235     void probarIngresarYMostrarProd(void)
236     {
237         tProd   prod;
238         int     cant = 0;
239
240         puts("Probando ingresar productos y mostrar productos.\n"
241              "======== ======== ========= = ======= =========");
242         if(ingresarProducto(&prod))
243             mostrarProducto(NULL);
244         do
245         {
246             mostrarProducto(&prod);
247             cant++;
248         } while(ingresarProducto(&prod));
249         fprintf(stdout, "Se mostraron %d productos.\n\n", cant);
250     }
251
252     void probarIngresarYMostrarTexto(void)
```

```
253 {
254     char    linea[90];
255     int     cant = 0;
256
257     puts("Probando ingresar lineas de texto mostrandolas.\n"
258          "======== ======== ====== == ===== ============");
259     while(ingresarTexto(linea, sizeof(linea)))
260     {
261         cant++;
262         printf("\"%s\"\n", linea);
263     }
264     fprintf(stdout, "Se mostraron %d lineas de texto.\n\n", cant);
265 }
266
267
268
269
270 void probarPonerYSacarDeLista(void)
271 {
272     tLista  lista;
273
274     crearLista(&lista);
275     if(listaLlena(&lista, 1))
276         puts("Que pasa? No hay lugar ni para un Byte? No hay memoria???");
277     if(listaVacia(&lista))
278         puts("Obviamente que la lista esta vacia! Recien creada!");
279     vaciarLista(&lista);
280 }
281
282
283 /* ------------------------------o---x---o------------------------------ */
284
```