```c
/* ---------------------------------o---x---o---------------------------------
 *          cola.h      TDA COLA con asignación dinámica de memoria
 * ---------------------------------o---x---o--------------------------------- */

#ifndef COLA_H_
#define COLA_H_

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


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

typedef struct
{
    tNodo  *pri,
           *ult;
} tCola;

void crearCola(tCola *p);

int  colaLlena(const tCola *p, unsigned cantBytes);

int  ponerEnCola(tCola *p, const void *d, unsigned cantBytes);

int  verPrimeroCola(const tCola *p, void *d, unsigned cantBytes);

int  colaVacia(const tCola *p);

int  sacarDeCola(tCola *p, void *d, unsigned cantBytes);

void vaciarCola(tCola *p);

#endif


/** ------------------------------------------------------------------------- **/
/* ---------------------------------o---x---o---------------------------------
 *          cola.c      TDA COLA con asignación dinámica de memoria
 * ---------------------------------o---x---o--------------------------------- */

#include "cola.h"

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

void crearCola(tCola *p)
{
    p->pri = NULL;
    p->ult = NULL;
}

int  colaLlena(const tCola *p, unsigned cantBytes)
{
    tNodo  *aux = (tNodo *)malloc(sizeof(tNodo));
    void   *info = malloc(cantBytes);
    free(aux);
    free(info);
    return aux == NULL || info == NULL;
}


int  ponerEnCola(tCola *p, const void *d, unsigned cantBytes)
{
    tNodo *nue = (tNodo *) malloc(sizeof(tNodo));

    if(nue == NULL || (nue->info = malloc(cantBytes)) == NULL)
    {
        free(nue);
        return 0;
    }
    memcpy(nue->info, d, cantBytes);
    nue->tamInfo = cantBytes;
    nue->sig = NULL;
    if(p->ult)
        p->ult->sig = nue;
    else
        p->pri = nue;
    p->ult = nue;
    return 1;
```

```
85  }
86
87
88  int  verPrimeroCola(const tCola *p, void *d, unsigned cantBytes)
89  {
90      if(p->pri == NULL)
91          return 0;
92      memcpy(d, p->pri->info, minimo(cantBytes, p->pri->tamInfo));
93      return 1;
94  }
95
96
97  int  colaVacia(const tCola *p)
98  {
99      return p->pri == NULL;
100 }
101
102
103 int  sacarDeCola(tCola *p, void *d, unsigned cantBytes)
104 {
105     tNodo *aux = p->pri;
106     if(aux == NULL)
107         return 0;
108     p->pri = aux->sig;
109     memcpy(d, aux->info, minimo(aux->tamInfo, cantBytes));
110     free(aux->info);
111     free(aux);
112     if(p->pri == NULL)
113         p->ult = NULL;
114     return 1;
115 }
116
117
118 void vaciarCola(tCola *p)
119 {
120     while(p->pri)
121     {
122         tNodo *aux = p->pri;
123         p->pri = aux->sig;
124         free(aux->info);
125         free(aux);
126     }
127     p->ult = NULL;
128 }
129
130
131 /** ------------------------------------------------------------------------ **/
132 /* ---------------------------------o---x---o---------------------------------
133  *          main.h     prueba del TDA COLA con asignación dinámica de memoria
134  * ---------------------------------o---x---o--------------------------------- */
135
136 #ifndef MAIN_H_
137 #define MAIN_H_
138
139 #include <stdio.h>
140
141
142 #include "../../productos/productos.h"
143 #include "../../lineasDeTexto/lineasTexto.h"
144 #include "../../colaDinamica/cola.h"
145
146
147 void probarIngresarYMostrarProd(void);
148
149 void probarIngresarYMostrarTexto(void);
150
151 void probarPonerYSacarDeCola(void);
152
153 #endif
154
155
156 /** ------------------------------------------------------------------------ **/
157 /* ---------------------------------o---x---o---------------------------------
158  *          main.c     prueba del TDA COLA con asignación dinámica de memoria
159  * ---------------------------------o---x---o--------------------------------- */
160
161 #include "main.h"
162
163
164 int main(void)
165 {
166     probarIngresarYMostrarProd();
167
168     probarIngresarYMostrarTexto();
```

```
169
170         probarPonerYSacarDeCola();
171
172         return 0;
173     }
174
175
176     void probarIngresarYMostrarProd(void)
177     {
178         tProd   prod;
179         int     cant = 0;
180
181         puts("Probando ingresar productos y mostrar productos.\n"
182              "======== ======== ========= = ======= =========");
183         if(ingresarProducto(&prod))
184             mostrarProducto(NULL);
185         do
186         {
187             mostrarProducto(&prod);
188             cant++;
189         } while(ingresarProducto(&prod));
190         fprintf(stdout, "Se mostraron %d productos.\n\n", cant);
191     }
192
193     void probarIngresarYMostrarTexto(void)
194     {
195         char    linea[90];
196         int     cant = 0;
197
198         puts("Probando ingresar lineas de texto mostrandolas.\n"
199              "======== ======== ====== == ===== ============");
200         while(ingresarTexto(linea, sizeof(linea)))
201         {
202             cant++;
203             printf("\"%s\"\n", linea);
204         }
205         fprintf(stdout, "Se mostraron %d lineas de texto.\n\n", cant);
206     }
207
208     int _probarLlenaYEncolar(tCola *cola)
209     {
210         tProd   prod;
211         int     cant = 0;
212
213         puts("Probando cola llena y poner en cola.");
214         mostrarProducto(NULL);
215         while(!colaLlena(cola, sizeof(prod)) && ingresarProducto(&prod))
216         {
217             if(!ponerEnCola(cola, &prod, sizeof(prod)))
218             {
219                 fprintf(stderr, "ERROR - inesperado - cola llena.\n");
220                 puts("no se pudo cargar la informacion y"
221                      " habria que tomar alguna decision drastica.");
222             }
223             mostrarProducto(&prod);
224             cant++;
225         }
226         return cant;
227     }
228
229
230     void _probarVerTope(tCola *cola)
231     {
232         tProd   prod;
233
234         puts("Probando ver el primero de la cola.");
235         if(verPrimeroCola(cola, &prod, sizeof(prod)))
236         {
237             mostrarProducto(NULL);
238             mostrarProducto(&prod);
239         }
240         else
241             puts("La cola estaba vacia.");
242         puts("");
243     }
244
245
246     void _probarVaciaYDesacolarN(tCola *cola, int canti)
247     {
248         tProd   prod;
249
250         printf("Probando cola vacia y sacar de cola %d productos (mostrandolos.\n",
251             canti);
252         if(colaVacia(cola))
```

```
253         puts("La cola esta vacia.");
254     else
255         mostrarProducto(NULL);
256     while(canti > 0 && sacarDeCola(cola, &prod, sizeof(prod)))
257     {
258         canti--;
259         mostrarProducto(&prod);
260     }
261     puts("");
262 }
263
264
265 int _probarVaciarColaYColaVacia(tCola *cola)
266 {
267     puts("Probando vaciar cola y cola vacia.");
268     vaciarCola(cola);
269     if(!colaVacia(cola))
270         return 0;  // fprintf(stderr, "ERROR - la cola debia estar vacia\n\n");
271     printf("Vaciar cola funciona!\n\n");
272     puts("");
273     return 1;
274 }
275
276
277 void _probarLlenaYEncolarTexto(tCola *cola)
278 {
279     char    linea[70];
280     int     cant = 0;
281
282     puts("Probando cola llena y poner en cola Texto.");
283     while(!colaLlena(cola, sizeof(linea)) &&
284         ingresarTexto(linea, sizeof(linea)))
285     {
286         if(!ponerEnCola(cola, linea, strlen(linea) + 1))
287         {
288             fprintf(stderr, "ERROR - inesperado - cola llena.\n");
289             puts("no se pudo cargar la informacion y"
290                 " habria que tomar alguna decision drastica.");
291         }
292         printf("\"%s\"\n", linea);
293         cant++;
294     }
295     printf("se pusieron %d lineas de texto en la cola.\n\n", cant);
296     printf("Probando sacar de cola con las lineas de texto.\n");
297 }
298
299
300 void _probarSacarDeColaTexto(tCola *cola)
301 {
302     char    linea[70];
303     int     cant = 0;
304
305     while(sacarDeCola(cola, linea, sizeof(linea)))
306     {
307         cant++;
308         printf("\"%s\"\n", linea);
309     }
310     printf("Se sacaron y mostraron %d lineas de texto\n\n", cant);
311 }
312
313
314 void probarPonerYSacarDeCola(void)
315 {
316     tCola   cola;
317     int     cant;
318
319     crearCola(&cola);
320
321     puts("Probando primitivas de cola con productos.\n"
322         "======== ========== == ==== === =========\n");
323     cant = _probarLlenaYEncolar(&cola);
324     printf("se pusieron %d productos en la cola.\n\n", cant);
325
326     _probarVerTope(&cola);
327
328     _probarVaciaYDesacolarN(&cola, cant - 2);
329
330     _probarVerTope(&cola);
331
332     if(_probarVaciarColaYColaVacia(&cola) != 1)
333         fprintf(stderr, "ERROR - inesperado, la cola NO QUEDO vacia\n");
334
335     puts("Probando primitivas de cola con lineas de texto.\n"
336         "======== ========== == ==== === ====== == =====\n");
```

```
337
338         _probarLlenaYEncolarTexto(&cola);
339
340         _probarSacarDeColaTexto(&cola);
341         cant = 0;
342
343         puts("ATENCION: se mostro el uso de una cola, en la que se pusieron"
344             " productos, se pro\n"
345             "baron todas las primitivas.  Una vez que se la dejo vacia "
346             "se encolaron lineas\n"
347             "de texto y luego se desencolaron.  Lo mas importante de esto "
348             "no es utilizar la\n"
349             "misma cola, lo que mas importa es que con las mismas primiti"
350             "vas se pueden enco-\n"
351             "lar distintos tipos de objetos, incluyendo lineas de texto d"
352             "e distinto tamano.\n");
353     }
354
355
356     /** ------------------------------------------------------------------------ **/
357
```