```c
/* ---------------------------------o---x---o---------------------------------
 *          pila.h     TDA PILA con asignación estática de memoria
 * ---------------------------------o---x---o--------------------------------- */

#ifndef PILA_H_
#define PILA_H_


#include <string.h>


#define     TAM_PILA        340

typedef struct
{
    char        pila[TAM_PILA];
    unsigned    tope;
} tPila;

void crearPila(tPila *p);
int  pilaLlena(const tPila *p, unsigned cantBytes);
int  ponerEnPila(tPila *p, const void *d, unsigned cantBytes);
int  verTope(const tPila *p, void *d, unsigned cantBytes);
int  pilaVacia(const tPila *p);
int  sacarDePila(tPila *p, void *d, unsigned cantBytes);
void vaciarPila(tPila *p);

#endif


/** ------------------------------------------------------------------------- **/
/* ---------------------------------o---x---o---------------------------------
 *          pila.c     TDA PILA con asignación estática de memoria
 * ---------------------------------o---x---o--------------------------------- */

#include "pila.h"

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


void crearPila(tPila *p)
{
    p->tope = TAM_PILA;
}

int  pilaLlena(const tPila *p, unsigned cantBytes)
{
    return p->tope < cantBytes + sizeof(unsigned);
}

int  ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
{
    if(p->tope < cantBytes + sizeof(unsigned))
        return 0;
    p->tope -= cantBytes;
    memcpy(p->pila + p->tope, d, cantBytes);
    p->tope -= sizeof(unsigned);
    memcpy(p->pila + p->tope, &cantBytes, sizeof(unsigned));
    return 1;
}

int  verTope(const tPila *p, void *d, unsigned cantBytes)
{
    unsigned tamInfo;

    if(p->tope == TAM_PILA)
        return 0;
    memcpy(&tamInfo, p->pila + p->tope, sizeof(unsigned));
    memcpy(d, p->pila + p->tope + sizeof(unsigned),
        minimo(cantBytes, tamInfo));
    return 1;
}

int  pilaVacia(const tPila *p)
{
    return p->tope == TAM_PILA;
}

int  sacarDePila(tPila *p, void *d, unsigned cantBytes)
{
    unsigned tamInfo;

    if(p->tope == TAM_PILA)
        return 0;
```

```
85      memcpy(&tamInfo, p->pila + p->tope, sizeof(unsigned));
86      p->tope += sizeof(unsigned);
87      memcpy(d, p->pila + p->tope, minimo(cantBytes, tamInfo));
88      p->tope += tamInfo;
89      return 1;
90  }
91
92  void vaciarPila(tPila *p)
93  {
94      p->tope = TAM_PILA;
95  }
96
97
98  /** ------------------------------------------------------------------------ **/
99  /* ---------------------------------o---x---o---------------------------------
100  *          main.h     prueba del TDA PILA con asignación estática de memoria
101  * ---------------------------------o---x---o--------------------------------- */
102
103 #ifndef MAIN_H_
104 #define MAIN_H_
105
106 #include <stdio.h>
107
108
109 #include "../../productos/productos.h"
110 #include "../../lineasDeTexto/lineasTexto.h"
111 #include "../../pilaEstatica/pila.h"
112
113
114 void probarIngresarYMostrarProd(void);
115
116 void probarIngresarYMostrarTexto(void);
117
118 void probarPonerYSacarDePila(void);
119
120
121 #endif // MAIN_H_
122
123
124 /** ------------------------------------------------------------------------ **/
125 /* ---------------------------------o---x---o---------------------------------
126  *          main.c     prueba del TDA PILA con asignación estatica de memoria
127  * ---------------------------------o---x---o--------------------------------- */
128
129 #include "main.h"
130
131
132 int main(void)
133 {
134     probarIngresarYMostrarProd();
135
136     probarIngresarYMostrarTexto();
137
138     probarPonerYSacarDePila();
139
140     return 0;
141 }
142
143
144 void probarIngresarYMostrarProd(void)
145 {
146     tProd   prod;
147     int     cant = 0;
148
149     puts("Probando ingresar productos y mostrar productos.\n"
150          "======== ======== ========= = ======= =========");
151     if(ingresarProducto(&prod))
152         mostrarProducto(NULL);
153     do
154     {
155         mostrarProducto(&prod);
156         cant++;
157     } while(ingresarProducto(&prod));
158     fprintf(stdout, "Se mostraron %d productos.\n\n", cant);
159 }
160
161 void probarIngresarYMostrarTexto(void)
162 {
163     char    linea[90];
164     int     cant = 0;
165
166     puts("Probando ingresar lineas de texto mostrandolas.\n"
167          "======== ======== ====== == ===== ============");
168     while(ingresarTexto(linea, sizeof(linea)))
```

```
169         {
170             cant++;
171             printf("\"%s\"\n", linea);
172         }
173         fprintf(stdout, "Se mostraron %d lineas de texto.\n\n", cant);
174     }
175
176
177     int _probarLlenaYApilar(tPila *pila)
178     {
179         tProd   prod;
180         int     cant = 0;
181
182         puts("Probando pila llena y poner en pila.");
183         mostrarProducto(NULL);
184         while(!pilaLlena(pila, sizeof(prod)) && ingresarProducto(&prod))
185         {
186             if(!ponerEnPila(pila, &prod, sizeof(prod)))
187             {
188                 fprintf(stderr, "ERROR - inesperado - pila llena.\n");
189                 puts("no se pudo cargar la informacion y"
190                      " habria que tomar alguna decision drastica.");
191             }
192             mostrarProducto(&prod);
193             cant++;
194         }
195         return cant;
196     }
197
198
199     void _probarVerTope(tPila *pila)
200     {
201         tProd   prod;
202
203         puts("Probando ver el tope de la pila.");
204         if(verTope(pila, &prod, sizeof(prod)))
205         {
206             mostrarProducto(NULL);
207             mostrarProducto(&prod);
208         }
209         else
210             puts("La pila estaba vacia.");
211         puts("");
212     }
213
214
215     void _probarVaciaYDesapilarN(tPila *pila, int canti)
216     {
217         tProd   prod;
218
219         printf("Probando pila vacia y sacar de pila %d productos (mostrandolos.\n",
220                canti);
221         if(pilaVacia(pila))
222             puts("La pila esta vacia.");
223         else
224             mostrarProducto(NULL);
225         while(canti > 0 && sacarDePila(pila, &prod, sizeof(prod)))
226         {
227             canti--;
228             mostrarProducto(&prod);
229         }
230         puts("");
231     }
232
233
234     int _probarVaciarPilaYPilaVacia(tPila *pila)
235     {
236         puts("Probando vaciar pila y pila vacia.");
237         vaciarPila(pila);
238         if(!pilaVacia(pila))
239             return 0;  // fprintf(stderr, "ERROR - la pila debia estar vacia\n\n");
240         printf("Vaciar pila funciona!\n\n");
241         puts("");
242         return 1;
243     }
244
245
246     void _probarLlenaYApilarTexto(tPila *pila)
247     {
248         char    linea[70];
249         int     cant = 0;
250
251         puts("Probando pila llena y poner en pila Texto.");
252         while(!pilaLlena(pila, sizeof(linea)) &&
```

```
253             ingresarTexto(linea, sizeof(linea)))
254     {
255         if(!ponerEnPila(pila, linea, strlen(linea) + 1))
256         {
257             fprintf(stderr, "ERROR - inesperado - pila llena.\n");
258             puts("no se pudo cargar la informacion y"
259                  " habria que tomar alguna decision drastica.");
260         }
261         printf("\"%s\"\n", linea);
262         cant++;
263     }
264     printf("se pusieron %d lineas de texto en la pila.\n\n", cant);
265     printf("Probando sacar de pila con las lineas de texto.\n");
266 }
267
268
269 void _probarSacarDePilaTexto(tPila *pila)
270 {
271     char    linea[70];
272     int     cant = 0;
273
274     while(sacarDePila(pila, linea, sizeof(linea)))
275     {
276         cant++;
277         printf("\"%s\"\n", linea);
278     }
279     printf("Se sacaron y mostraron %d lineas de texto\n\n", cant);
280 }
281
282
283 void probarPonerYSacarDePila(void)
284 {
285     tPila   pila;
286     int     cant;
287
288     crearPila(&pila);
289
290     puts("Probando primitivas de pila con productos.\n"
291          "======== ========== == ==== === =========\n");
292     cant = _probarLlenaYApilar(&pila);
293     printf("se pusieron %d productos en la pila.\n\n", cant);
294
295     _probarVerTope(&pila);
296
297     _probarVaciaYDesapilarN(&pila, cant - 2);
298
299     _probarVerTope(&pila);
300
301     if(_probarVaciarPilaYPilaVacia(&pila) != 1)
302         fprintf(stderr, "ERROR - inesperado, la pila NO QUEDO vacia\n");
303
304     puts("Probando primitivas de pila con lineas de texto.\n"
305          "======== ========== == ==== === ====== == =====\n");
306
307     _probarLlenaYApilarTexto(&pila);
308
309     _probarSacarDePilaTexto(&pila);
310     cant = 0;
311
312     puts("ATENCION: se mostro el uso de una pila, en la que se apilaron"
313          " productos, se pro\n"
314          "baron todas las primitivas.  Una vez que se la dejo vacia "
315          "se apilaron lineas\n"
316          "de texto y luego se desapilaron.  Lo mas importante de esto "
317          "no es utilizar la\n"
318          "misma pila, lo que mas importa es que con las mismas primiti"
319          "vas se pueden api-\n"
320          "lar distintos tipos de objetos, incluyendo lineas de texto d"
321          "e distinto tamano.\n");
322 }
323
324
325 /** ------------------------------------------------------------------- **/
326
```