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

#ifndef PILA_H_
#define PILA_H_

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


typedef struct sNodo
{
    void           *info;
    unsigned        tamInfo;
    struct sNodo   *sig;
} tNodo;
typedef tNodo *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 dinámica de memoria
 * ------------------------------o---x---o------------------------------ */

#include "pila.h"


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


void crearPila(tPila *p)
{
    *p = NULL;
}

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

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

int  ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
{
    tNodo *nue;

    if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
    (nue->info = malloc(cantBytes)) == NULL)
    {
        free(nue);
        return 0;
    }
    memcpy(nue->info, d, cantBytes);
    nue->tamInfo = cantBytes;
    nue->sig = *p;
    *p = nue;
    return 1;
}

int  verTope(const tPila *p, void *d, unsigned cantBytes)
{
    if(*p == NULL)
        return 0;
    memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
    return 1;
}

int  pilaVacia(const tPila *p)
{
    return *p == NULL;
```

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

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

```
253         printf("Vaciar pila funciona!\n\n");
254         puts("");
255         return 1;
256     }
257
258
259     void _probarLlenaYApilarTexto(tPila *pila)
260     {
261         char    linea[70];
262         int     cant = 0;
263
264         puts("Probando pila llena y poner en pila Texto.");
265         while(!pilaLlena(pila, sizeof(linea)) &&
266               ingresarTexto(linea, sizeof(linea)))
267         {
268             if(!ponerEnPila(pila, linea, strlen(linea) + 1))
269             {
270                 fprintf(stderr, "ERROR - inesperado - pila llena.\n");
271                 puts("no se pudo cargar la informacion y"
272                      " habria que tomar alguna decision drastica.");
273             }
274             printf("\"%s\"\n", linea);
275             cant++;
276         }
277         printf("se pusieron %d lineas de texto en la pila.\n\n", cant);
278         printf("Probando sacar de pila con las lineas de texto.\n");
279     }
280
281
282     void _probarSacarDePilaTexto(tPila *pila)
283     {
284         char    linea[70];
285         int     cant = 0;
286
287         while(sacarDePila(pila, linea, sizeof(linea)))
288         {
289             cant++;
290             printf("\"%s\"\n", linea);
291         }
292         printf("Se sacaron y mostraron %d lineas de texto\n\n", cant);
293     }
294
295
296     void probarPonerYSacarDePila(void)
297     {
298         tPila   pila;
299         int     cant;
300
301         crearPila(&pila);
302
303         puts("Probando primitivas de pila con productos.\n"
304              "======== ========== == ==== === =========\n");
305         cant = _probarLlenaYApilar(&pila);
306         printf("se pusieron %d productos en la pila.\n\n", cant);
307
308         _probarVerTope(&pila);
309
310         _probarVaciaYDesapilarN(&pila, cant - 2);
311
312         _probarVerTope(&pila);
313
314         if(_probarVaciarPilaYPilaVacia(&pila) != 1)
315             fprintf(stderr, "ERROR - inesperado, la pila NO QUEDO vacia\n");
316
317         puts("Probando primitivas de pila con lineas de texto.\n"
318              "======== ========== == ==== === ====== == =====\n");
319
320         _probarLlenaYApilarTexto(&pila);
321
322         _probarSacarDePilaTexto(&pila);
323         cant = 0;
324
325         puts("ATENCION: se mostro el uso de una pila, en la que se apilaron"
326              " productos, se pro\n"
327              "baron todas las primitivas.  Una vez que se la dejo vacia "
328              "se apilaron lineas\n"
329              "de texto y luego se desapilaron.  Lo mas importante de esto "
330              "no es utilizar la\n"
331              "misma pila, lo que mas importa es que con las mismas primiti"
332              "vas se pueden api-\n"
333              "lar distintos tipos de objetos, incluyendo lineas de texto d"
334              "e distinto tamano.\n");
335     }
336
```

```
337
338     /** ------------------------------------------------------------------------ **/
339
```