

```

1  /*-----o--x--o-----CRLF
2  *.-----main.hCRLF
3  /*-----o--x--o-----*/CRLF
4  #ifndef MAIN_H_CRLF
5  #define MAIN_H_CRLF
6  CRLF
7  #include <stdio.h>CRLF
8  CRLF
9  #include "productos/productos.h"CRLF
10 #include "pilaEstatica/pila.h"CRLF
11 #include "pilaDinamica/pila.h"CRLF
12 CRLF
13 CRLF
14 CRLF
15 void probarIngresarYMostrarProd(void);CRLF
16 CRLF
17 void probarPonerYSacarDePila(void);CRLF
18 CRLF
19 #endifCRLF
20 /*-----o--x--o-----CRLF
21 *.-----main.cCRLF
22 /*-----o--x--o-----*/CRLF
23 #include "main.h"CRLF
24 CRLF
25 CRLF
26 int main(void)CRLF
27 {CRLF
28     printf("Tamano tProd: %d\n", sizeof(tProd));CRLF
29     probarIngresarYMostrarProd();CRLF
30     CRLF
31     probarPonerYSacarDePila();CRLF
32     CRLF
33     return 0;CRLF
34 }CRLF
35 CRLF
36 CRLF
37 void probarIngresarYMostrarProd(void)CRLF
38 {CRLF
39     tProd prod;CRLF
40     int result;CRLF
41     cant = 0;CRLF
42     CRLF
43     puts("Probando ingresar productos y mostrar productos");CRLF
44     result = ingresarProducto(&prod);CRLF
45     if(result)CRLF
46         mostrarProducto(NULL);CRLF
47     while(result)CRLF
48     {CRLF
49         mostrarProducto(&prod);CRLF
50         result = ingresarProducto(&prod);CRLF
51         cant++;CRLF
52     }CRLF
53     fprintf(stdout, "Se mostraron %d productos.\n\n", cant);CRLF
54 }CRLF
55 CRLF
56 void probarPonerYSacarDePila(void)CRLF
57 {CRLF
58     tProd prod;CRLF
59     tPila pila;CRLF
60     int result;CRLF
61     llena;CRLF
62     CRLF
63     crearPila(&pila);CRLF
64     llena = pilaLlena(&pila, sizeof(prod));CRLF
65     if(!llenar)CRLF
66     {CRLF
67         result = ingresarProducto(&prod);CRLF
68         puts("Procediendo a poner en pila");CRLF
69         mostrarProducto(NULL);CRLF
70     }CRLF
71     while(result && !llenar)CRLF
72     {CRLF
73     CRLF

```

```

75 .....if(!ponerEnPila(&pila, &prod, sizeof(prod)))CRLF
76 .....fprintf(stderr, "ERROR -- inesperado: pila llena\n");CRLF
77 .....puts("no se pudo cargar la informacion");CRLF
78 .....}CRLF
79 .....mostrarProducto(&prod);CRLF
80 .....llena = pilaLlena(&pila, sizeof(prod));CRLF
81 .....if(!llenar)CRLF
82 .....result = ingresarProducto(&prod);CRLF
83 .....elseCRLF
84 .....puts("Se lleno la pila");CRLF
85 .....}CRLF
86 CRLF
87 .....puts("\nMostrando el tope de la pila");CRLF
88 .....if(!pilaVacía(&pila))CRLF
89 .....{CRLF
90 .....tProd otro;CRLF
91 .....verTope(&pila, &otro, sizeof(otro));CRLF
92 .....mostrarProducto(&otro);CRLF
93 .....}CRLF
94 .....elseCRLF
95 .....puts("La pila estaba vacía");CRLF
96 CRLF
97 .....puts("\nProcediendo a sacar de la pila y mostrar");CRLF
98 .....if(pilaVacía(&pila))CRLF
99 .....puts("La pila está vacía");CRLF
100 .....elseCRLF
101 .....mostrarProducto(NULL);CRLF
102 .....while(sacarDePila(&pila, &prod, sizeof(prod)))CRLF
103 .....mostrarProducto(&prod);CRLF
104 .....puts("");CRLF
105 }CRLF
106 CRLF
107 /*-----o---x---o-----CRLF
108 *.....productos.hCRLF
109 *-----o---x---o-----*/CRLF
110 #ifndef PRODUCTOS_H CRLF
111 #define PRODUCTOS_H CRLF
112 CRLF
113 #include <stdio.h>CRLF
114 CRLF
115 CRLF
116 typedef struct CRLF
117 {CRLF
118 .....char codProd[11], CRLF
119 .....descrip[46];CRLF
120 } tProd;CRLF
121 CRLF
122 int ingresarProducto(tProd *d);CRLF
123 CRLF
124 void mostrarProducto(const tProd *d);CRLF
125 CRLF
126 CRLF
127 #endifCRLF
128 /*-----o---x---o-----CRLF
129 *.....productos.cCRLF
130 *-----o---x---o-----*/CRLF
131 #include "productos.h"CRLF
132 CRLF
133 CRLF
134 int ingresarProducto(tProd *d)CRLF
135 {CRLF
136 .....static const tProd productos[] = {CRLF
137 .....//1234567890...123456789 123456789 123456789 123456789 123456789CRLF
138 .....{"clavoro3/4", "Clavo de oro 24 kilates de 3/4 de pulgada"},CRLF
139 .....{"martillo3K", "Martillo bolita con saca clavos de 3 kilos"},CRLF
140 .....{"alameso1", "Alambre de yeso de un milimetro de espesor"},CRLF
141 .....{"rem-vid15", "Remache de vidrio de 1,5 milímetros"},CRLF
142 .....{"plom-telgo", "Plomada de poliestireno expandido"},CRLF
143 .....{"limagoma17", "Lima de goma de 17 pulgadas"} };CRLF
144 .....static int posi = 0;CRLF
145 CRLF
146 .....if(posi == sizeof(productos) / sizeof(tProd))CRLF

```

```

147     ....{CRLF
148     .....posi = 0;CRLF
149     .....return 0;CRLF
150     ....}CRLF
151     ....*d = productos[posi];CRLF
152     ....posi++;CRLF
153     CRLF
154     ....return 1;CRLF
155     }CRLF
156     CRLF
157     void mostrarProducto(const tProd *d)CRLF
158     {CRLF
159     ....if(d)CRLF
160     .....fprintf(stdout,CRLF
161     .....    "%-*s %-*s...\n",CRLF
162     .....    sizeof(d->codProd) - 1, d->codProd,CRLF
163     .....    sizeof(d->descrip) - 1, d->descrip);CRLF
164     ....elseCRLF
165     .....fprintf(stdout,CRLF
166     .....    "%-*s %-*s...\n",CRLF
167     .....    sizeof(d->codProd) - 1, sizeof(d->codProd) - 1,CRLF
168     .....    "Cod. Producto",CRLF
169     .....    sizeof(d->descrip) - 1, sizeof(d->descrip) - 1,CRLF
170     .....    "Descripcion del producto");CRLF
171     }CRLF
172     CRLF
173     /*-----o--x--o-----CRLF
174     *.....pila.h.....ESTÁTICA CRLF
175     *-----o--x--o-----*/CRLF
176     #ifdef ESTATICA CRLF
177     CRLF
178     #ifndef PILA_H CRLF
179     #define PILA_H CRLF
180     CRLF
181     ///pila ESTÁTICA CRLF
182     CRLF
183     #include <string.h>CRLF
184     CRLF
185     #define minimo(X, Y) ((X) <= (Y) ? (X) : (Y))CRLF
186     CRLF
187     CRLF
188     #define TAM_PILA 340CRLF
189     CRLF
190     typedef struct CRLF
191     {CRLF
192     ....char pila[TAM_PILA];CRLF
193     ....unsigned tope;CRLF
194     } tPila;CRLF
195     CRLF
196     void crearPila(tPila *p);CRLF
197     int pilaLlena(const tPila *p, unsigned cantBytes);CRLF
198     int ponerEnPila(tPila *p, const void *d, unsigned cantBytes);CRLF
199     int verTope(const tPila *p, void *d, unsigned cantBytes);CRLF
200     int pilaVacia(const tPila *p);CRLF
201     int sacarDePila(tPila *p, void *d, unsigned cantBytes);CRLF
202     void vaciarPila(tPila *p);CRLF
203     CRLF
204     #endifCRLF
205     CRLF
206     #endifCRLF
207     /*-----o--x--o-----CRLF
208     *.....pila.c.....ESTÁTICA CRLF
209     *-----o--x--o-----*/CRLF
210     #ifdef ESTATICA CRLF
211     CRLF
212     ///pila ESTÁTICA CRLF
213     CRLF
214     #include "pila.h" CRLF
215     CRLF
216     CRLF
217     void crearPila(tPila *p)CRLF
218     {CRLF
219     ....p->tope = TAM_PILA;CRLF

```

```

220 }CRLF
221 CRLF
222 int pilaLlena(const tPila *p, unsigned cantBytes)CRLF
223 {CRLF
224     return p->tope < cantBytes + sizeof(unsigned);CRLF
225 }CRLF
226 CRLF
227 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)CRLF
228 {CRLF
229     if(p->tope < cantBytes + sizeof(unsigned))CRLF
230         return 0;CRLF
231     p->tope -= cantBytes;CRLF
232     memcpy(p->pila + p->tope, d, cantBytes);CRLF
233     p->tope -= sizeof(unsigned);CRLF
234     memcpy(p->pila + p->tope, &cantBytes, sizeof(unsigned));CRLF
235     return 1;CRLF
236 }CRLF
237 CRLF
238 int verTope(const tPila *p, void *d, unsigned cantBytes)CRLF
239 {CRLF
240     unsigned tamInfo;CRLF
241     CRLF
242     if(p->tope == TAM_PILA)CRLF
243         return 0;CRLF
244     memcpy(&tamInfo, p->pila + p->tope, sizeof(unsigned));CRLF
245     memcpy(d, p->pila + p->tope + sizeof(unsigned), CRLF
246         minimo(cantBytes, tamInfo));CRLF
247     return 1;CRLF
248 }CRLF
249 CRLF
250 int pilaVacía(const tPila *p)CRLF
251 {CRLF
252     return p->tope == TAM_PILA;CRLF
253 }CRLF
254 CRLF
255 int sacarDePila(tPila *p, void *d, unsigned cantBytes)CRLF
256 {CRLF
257     unsigned tamInfo;CRLF
258     CRLF
259     if(p->tope == TAM_PILA)CRLF
260         return 0;CRLF
261     memcpy(&tamInfo, p->pila + p->tope, sizeof(unsigned));CRLF
262     p->tope += sizeof(unsigned);CRLF
263     memcpy(d, p->pila + p->tope, minimo(cantBytes, tamInfo));CRLF
264     p->tope += tamInfo;CRLF
265     return 1;CRLF
266 }CRLF
267 CRLF
268 void vaciarPila(tPila *p)CRLF
269 {CRLF
270     p->tope = TAM_PILA;CRLF
271 }CRLF
272 CRLF
273 #endifCRLF
274 CRLF
275 /*-----o--x--o-----CRLF
276 *      pila.h      DINÁMICA CRLF
277 *-----o--x--o-----*/CRLF
278 #ifdef DINAMICA CRLF
279 CRLF
280 #ifndef PILA_H CRLF
281 #define PILA_H CRLF
282 CRLF
283 ///-pila- DINÁMICA CRLF
284 #include <stdlib.h>CRLF
285 #include <string.h>CRLF
286 CRLF
287 #define minimo(X, Y) ((X) <= (Y) ? (X) : (Y))CRLF
288 CRLF
289 typedef struct sNodoCRLF
290 {CRLF
291     void *info;CRLF
292     unsigned tamInfo;CRLF

```

```

293     ....struct sNodo... *sig;CRLF
294 } tNodo;CRLF
295 typedef tNodo *tPila;CRLF
296 CRLF
297 void crearPila(tPila *p);CRLF
298 int pilaLlena(const tPila *p, unsigned cantBytes);CRLF
299 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes);CRLF
300 int verTope(const tPila *p, void *d, unsigned cantBytes);CRLF
301 int pilaVacía(const tPila *p);CRLF
302 int sacarDePila(tPila *p, void *d, unsigned cantBytes);CRLF
303 void vaciarPila(tPila *p);CRLF
304 CRLF
305 #endifCRLF
306 CRLF
307 #endifCRLF
308 /*-----o---x---o-----CRLF
309 *.....pila.c.....DINÁMICA CRLF
310 *-----o---x---o-----*/CRLF
311 #ifdef DINAMICA CRLF
312 CRLF
313 CRLF
314 ///pila DINÁMICA CRLF
315 CRLF
316 #include "pila.h" CRLF
317 CRLF
318 void crearPila(tPila *p) CRLF
319 { CRLF
320     ....*p = NULL;CRLF
321 } CRLF
322 CRLF
323 int pilaLlena(const tPila *p, unsigned cantBytes) CRLF
324 { CRLF
325     ....tNodo *aux = (tNodo *) malloc(sizeof(tNodo));CRLF
326     ....void *info = malloc(cantBytes);CRLF
327 CRLF
328     ....free(aux);CRLF
329     ....free(info);CRLF
330     ....return aux == NULL || info == NULL;CRLF
331 } CRLF
332 CRLF
333 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes) CRLF
334 { CRLF
335     ....tNodo *nue;CRLF
336 CRLF
337     ....if((nue = (tNodo *) malloc(sizeof(tNodo))) == NULL ||CRLF
338     ....(nue->info = malloc(cantBytes)) == NULL) CRLF
339     ....{CRLF
340     ....    ....free(nue);CRLF
341     ....    ....return 0;CRLF
342     ....} CRLF
343     ....memcpy(nue->info, d, cantBytes);CRLF
344     ....nue->tamInfo = cantBytes;CRLF
345     ....nue->sig = *p;CRLF
346     ....*p = nue;CRLF
347     ....return 1;CRLF
348 } CRLF
349 CRLF
350 int verTope(const tPila *p, void *d, unsigned cantBytes) CRLF
351 { CRLF
352     ....if(*p == NULL) CRLF
353     ....    ....return 0;CRLF
354     ....memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));CRLF
355     ....return 1;CRLF
356 } CRLF
357 CRLF
358 int pilaVacía(const tPila *p) CRLF
359 { CRLF
360     ....return *p == NULL;CRLF
361 } CRLF
362 CRLF
363 int sacarDePila(tPila *p, void *d, unsigned cantBytes) CRLF
364 { CRLF
365     ....tNodo *aux = *p;CRLF

```

```
366     CRLF
367     ....if(aux == NULL) CRLF
368     .....return 0; CRLF
369     ....*p = aux->sig; CRLF
370     ....memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo)); CRLF
371     ....free(aux->info); CRLF
372     ....free(aux); CRLF
373     ....return 1; CRLF
374 } CRLF
375 CRLF
376 void vaciarPila(tPila *p) CRLF
377 { CRLF
378     ....while(*p) CRLF
379     ....{ CRLF
380     .....tNodo *aux = *p; CRLF
381     CRLF
382     .....*p = aux->sig; CRLF
383     .....free(aux->info); CRLF
384     .....free(aux); CRLF
385     ....} CRLF
386 } CRLF
387 CRLF
388 #endif CRLF
389 CRLF
390
```