

```

1  /*-----o--x--o-----CRLE
2  *-----main.hCRLE
3  *-----o--x--o-----*/CRLE
4  #ifndef MAIN_H_CRLE
5  #define MAIN_H_CRLE
6  CRLE
7  #include <stdio.h>CRLE
8  CRLE
9  #include "productos/productos.h"CRLE
10 #include "lista/lista.h"CRLE
11 CRLE
12 CRLE
13 void probarIngresarYMostrarProd(void);CRLE
14 CRLE
15 void probarPonerAdelanteVerUltimoVerPrimeroYVaciar(void);CRLE
16 CRLE
17 void probarPonerAtrasSacarUltimoSacarPrimeroYMostrar(void);CRLE
18 CRLE
19 void probarPonerEnOrdenMostrarAlRevesYOrdenar(void);CRLE
20 CRLE
21 CRLE
22 #endifCRLE
23 /*-----o--x--o-----CRLE
24 *-----main.cCRLE
25 *-----o--x--o-----*/CRLE
26 CRLE
27 #include "main.h"CRLE
28 CRLE
29 CRLE
30 int main(void)CRLE
31 {CRLE
32     probarIngresarYMostrarProd();CRLE
33     CRLE
34     probarPonerAdelanteVerUltimoVerPrimeroYVaciar();CRLE
35     CRLE
36     probarPonerAtrasSacarUltimoSacarPrimeroYMostrar();CRLE
37     CRLE
38     probarPonerEnOrdenMostrarAlRevesYOrdenar();CRLE
39     CRLE
40     return 0;CRLE
41 }CRLE
42 CRLE
43 CRLE
44 void probarIngresarYMostrarProd(void)CRLE
45 {CRLE
46     tProd prod;CRLE
47     int result;CRLE
48     cant = 0;CRLE
49     CRLE
50     puts("\nProbando ingresar productos y mostrar productos");CRLE
51     result = ingresarProducto(&prod);CRLE
52     if(result)CRLE
53         mostrarProducto(NULL);CRLE
54     while(result)CRLE
55     {CRLE
56         mostrarProducto(&prod);CRLE
57         result = ingresarProducto(&prod);CRLE
58         cant++;CRLE
59     }CRLE
60     printf(stdout, "Se mostraron %d productos.\n", cant);CRLE
61 }CRLE
62 CRLE
63 void probarPonerAdelanteVerUltimoVerPrimeroYVaciar(void)CRLE
64 {CRLE
65     tLista lista;CRLE
66     tProd prod;CRLE
67     int cant = 0;CRLE
68     CRLE
69     puts("\nFUNCION: probarPonerAdelanteVerUltimoVerPrimeroYVaciar");CRLE

```

```

70     crearLista(&lista); CRLE
71 CRLE
72     printf("Insertando al comienzo de la lista\n"); CRLE
73     while(ingresarProducto(&prod) && CRLE
74         ponerAlComienzo(&lista, &prod, sizeof(tProd))) CRLE
75     { CRLE
76         if(cant == 0) CRLE
77             mostrarProducto(NULL); CRLE
78             mostrarProducto(&prod); CRLE
79             cant++; CRLE
80     } CRLE
81     printf("Se insertaron %d productos en la lista\n", cant); CRLE
82     if(verUltimoLista(&lista, &prod, sizeof(prod))) CRLE
83     { CRLE
84         printf("Ultimo de la lista\n"); CRLE
85         mostrarProducto(NULL); CRLE
86         mostrarProducto(&prod); CRLE
87     } CRLE
88     if(verPrimeroLista(&lista, &prod, sizeof(prod))) CRLE
89     { CRLE
90         printf("Primero de la lista\n"); CRLE
91         mostrarProducto(NULL); CRLE
92         mostrarProducto(&prod); CRLE
93     } CRLE
94     vaciarLista(&lista); CRLE
95 } CRLE
96 CRLE
97 CRLE
98 void probarPonerAtrasSacarUltimoSacarPrimeroYMostrar(void) CRLE
99 { CRLE
100     tLista lista; CRLE
101     tProd prod; CRLE
102     int cant = 0; CRLE
103 CRLE
104     puts("\nFUNCION: probarPonerAtrasSacarUltimoSacarPrimeroYMostrar"); CRLE
105     crearLista(&lista); CRLE
106 CRLE
107     printf("Insertando al final de la lista\n"); CRLE
108     while(ingresarProducto(&prod) && CRLE
109         ponerAlFinal(&lista, &prod, sizeof(tProd))) CRLE
110     { CRLE
111         if(cant == 0) CRLE
112             mostrarProducto(NULL); CRLE
113             mostrarProducto(&prod); CRLE
114             cant++; CRLE
115     } CRLE
116     printf("Se insertaron %d productos en la lista\n", cant); CRLE
117     if(sacarPrimeroLista(&lista, &prod, sizeof(prod))) CRLE
118     { CRLE
119         printf("Primero de la lista\n"); CRLE
120         mostrarProducto(NULL); CRLE
121         mostrarProducto(&prod); CRLE
122     } CRLE
123     if(sacarUltimoLista(&lista, &prod, sizeof(prod))) CRLE
124     { CRLE
125         printf("Ultimo de la lista\n"); CRLE
126         mostrarProducto(NULL); CRLE
127         mostrarProducto(&prod); CRLE
128     } CRLE
129     cant = mostrarLista(&lista, mostrarProducto); CRLE
130     if(cant) CRLE
131         printf("Se mostraron %d productos\n", cant); CRLE
132     vaciarLista(&lista); CRLE
133 } CRLE
134 CRLE
135 void probarPonerEnOrdenMostrarAlRevesYOrdenar(void) CRLE
136 { CRLE
137     tLista lista; CRLE
138     tProd prod; CRLE

```

```

139     int cant = 0;
140     int dupl = 0;
141
142     puts("\nFUNCION: probarPonerEnOrdenMostrarAlRevesYOrdenar");
143     crearLista(&lista);
144
145     printf("Insertando en orden por clave del producto\n");
146     while(ingresarProducto(&prod))
147     {
148         int result = ponerEnOrdenOAcumular(&lista, &prod, sizeof(tProd),
149                                           compararProdXCodProd, NULL);
150         if(cant == 0)
151             mostrarProducto(NULL);
152         mostrarProducto(&prod);
153         if(result == TODO_BIEN)
154             cant++;
155         if(result == CLA_DUP)
156             dupl++;
157     }
158     printf("Se insertaron %d productos en la lista y hubo %d duplicados\n",
159           cant, dupl);
160     cant = mostrarListaAlReves(&lista, mostrarProducto);
161     if(cant)
162         printf("Se mostraron %d productos\n", cant);
163     puts("Ordenando por descripcion del producto");
164     ordenarLista(&lista, compararProdXDescrip);
165     cant = mostrarListaAlReves(&lista, mostrarProducto);
166     if(cant)
167         printf("Se mostraron %d productos\n", cant);
168     vaciarLista(&lista);
169 }
170
171 /*-----o---x---o-----
172 *-----productos.h
173 *-----o---x---o-----*/
174
175 #ifndef PRODUCTOS_H
176 #define PRODUCTOS_H
177
178 #include <stdio.h>
179 #include <string.h>
180
181 typedef struct
182 {
183     char codProd[11],
184     descrip[46];
185 } tProd;
186
187 int ingresarProducto(tProd *d);
188 void mostrarProducto(const void *d);
189 int compararProdXCodProd(const void *d1, const void *d2);
190 int compararProdXDescrip(const void *d1, const void *d2);
191
192 #endif
193
194 /*-----o---x---o-----
195 *-----productos.c
196 *-----o---x---o-----*/
197
198 #include "productos.h"
199
200 int ingresarProducto(tProd *d)
201 {
202     static const tProd productos[] = {

```

```

208 //1234567890...123456789...123456789...123456789...123456789...CRLF
209 .....{"clavoro3/4","Clavo de oro 24 kilates de 3/4 de pulgada"},CRLF
210 .....{"martillo3K","Martillo bolita con saca clavos de 3 kilos"},CRLF
211 .....{"yesoalam1","Alambre de yeso de un milimetro de espesor"},CRLF
212 .....{"vidrem-15","Remache de vidrio de 1,5 milímetros"},CRLF
213 .....{"plom-telgo","Plomada de poliestireno expandido"},CRLF
214 .....{"limagoma17","Lima de goma de 17 pulgadas"};};CRLF
215 static int posi = 0;CRLF
216 CRLF
217 ....if(posi == sizeof(productos) / sizeof(tProd))CRLF
218 .....{CRLF
219 .....posi = 0;CRLF
220 .....return 0;CRLF
221 .....}CRLF
222 ....*d = productos[posi];CRLF
223 ....posi++;CRLF
224 CRLF
225 ....return 1;CRLF
226 }CRLF
227 CRLF
228 void mostrarProducto(const void *d)CRLF
229 {CRLF
230 ....tProd *dProd = (tProd *)d;CRLF
231 ....if(d)CRLF
232 .....fprintf(stdout,CRLF
233 .....    "%-*s%-*s...\n",CRLF
234 .....    sizeof(dProd->codProd)-1,dProd->codProd,CRLF
235 .....    sizeof(dProd->descrip)-1,dProd->descrip);CRLF
236 ....elseCRLF
237 .....fprintf(stdout,CRLF
238 .....    "%-*.*s%-*.*s...\n",CRLF
239 .....    sizeof(dProd->codProd)-1,sizeof(dProd->codProd)-1,CRLF
240 .....    "Cod. Producto",CRLF
241 .....    sizeof(dProd->descrip)-1,sizeof(dProd->descrip)-1,CRLF
242 .....    "Descripcion del producto");CRLF
243 }CRLF
244 CRLF
245 int compararProdXCodProd(const void *d1,const void *d2)CRLF
246 {CRLF
247 ....tProd *p = (tProd *)d1,CRLF
248 ....q = (tProd *)d2;CRLF
249 CRLF
250 ....return strcmpi(p->codProd,q->codProd);CRLF
251 }CRLF
252 CRLF
253 int compararProdXDescrip(const void *d1,const void *d2)CRLF
254 {CRLF
255 ....tProd *p = (tProd *)d1,CRLF
256 ....q = (tProd *)d2;CRLF
257 CRLF
258 ....return strcmpi(p->descrip,q->descrip);CRLF
259 }CRLF
260 CRLF
261 /*-----o---x---o-----CRLF
262 .....lista.hCRLF
263 .....o---x---o-----*/CRLF
264 #ifndef LISTA_H_CRLF
265 #define LISTA_H_CRLF
266 CRLF
267 #include <stdlib.h>CRLF
268 #include <string.h>CRLF
269 CRLF
270 #define SIN_MEM.....1CRLF
271 #define CLA_DUP.....2CRLF
272 #define TODO_BIEN.....0CRLF
273 CRLF
274 #define minimo(X,Y) ((X)<=(Y)?(X):(Y))CRLF
275 CRLF
276 typedef struct sNodoCRLF

```

```

277 {CRLF
278     ...void ...*info;CRLF
279     ...unsigned ...tamInfo;CRLF
280     ...struct sNodo ...*sig;CRLF
281 } tNodo;CRLF
282 typedef tNodo *tLista;CRLF
283 CRLF
284 CRLF
285 void crearLista (tLista *p);CRLF
286 CRLF
287 int listaVacia (const tLista *p);CRLF
288 CRLF
289 int listaLlena (const tLista *p, unsigned cantBytes);CRLF
290 CRLF
291 void vaciarLista (tLista *p);CRLF
292 CRLF
293 int ponerAlComienzo (tLista *p, const void *d, unsigned cantBytes);CRLF
294 CRLF
295 CRLF
296 int sacarPrimeroLista (tLista *p, void *d, unsigned cantBytes);CRLF
297 CRLF
298 int verPrimeroLista (const tLista *p, void *d, unsigned cantBytes);CRLF
299 CRLF
300 int ponerAlFinal (tLista *p, const void *d, unsigned cantBytes);CRLF
301 CRLF
302 int sacarUltimoLista (tLista *p, void *d, unsigned cantBytes);CRLF
303 CRLF
304 int verUltimoLista (const tLista *p, void *d, unsigned cantBytes);CRLF
305 CRLF
306 CRLF
307 int mostrarLista (const tLista *p, void (*mostrar) (const void *));CRLF
308 CRLF
309 int mostrarListaAlReves (const tLista *p, void (*mostrar) (const void *));CRLF
310 CRLF
311 void ordenarLista (tLista *p, int (*comparar) (const void *, const void *));CRLF
312 CRLF
313 int ponerEnOrdenOAcumular (tLista *p, const void *d, unsigned cantBytes,CRLF
314     ...int (*comparar) (const void *, const void *),CRLF
315     ...void (*acumular) (void **, const void *,CRLF
316     ...unsigned *, unsigned));CRLF
317 CRLF
318 CRLF
319 #endifCRLF
320 /*-----o--x--o-----CRLF
321 *-----lista.cCRLF
322 *-----o--x--o-----*/CRLF
323 CRLF
324 #include "lista.h"CRLF
325 CRLF
326 void crearLista (tLista *p) CRLF
327 { CRLF
328     ...*p = NULL;CRLF
329 } CRLF
330 CRLF
331 int listaVacia (const tLista *p) CRLF
332 { CRLF
333     ...return *p == NULL;CRLF
334 } CRLF
335 CRLF
336 int listaLlena (const tLista *p, unsigned cantBytes) CRLF
337 { CRLF
338     ...tNodo *aux = (tNodo *) malloc (sizeof (tNodo));CRLF
339     ...void *info = malloc (cantBytes);CRLF
340     CRLF
341     ...free (aux);CRLF
342     ...free (info);CRLF
343     ...return aux == NULL || info == NULL;CRLF
344 } CRLF
345 CRLF

```

```

346 void vaciarLista (tLista *p) CRLE
347 { CRLE
348     while (*p) CRLE
349     { CRLE
350         tNodo *aux = *p; CRLE
351         CRLE
352         *p = aux->sig; CRLE
353         free (aux->info); CRLE
354         free (aux); CRLE
355     } CRLE
356 } CRLE
357 CRLE
358 int ponerAlComienzo (tLista *p, const void *d, unsigned cantBytes) CRLE
359 { CRLE
360     tNodo *nue; CRLE
361     CRLE
362     if ((nue = (tNodo *) malloc (sizeof (tNodo))) == NULL || CRLE
363         (nue->info = malloc (cantBytes)) == NULL) CRLE
364     { CRLE
365         free (nue); CRLE
366         return 0; CRLE
367     } CRLE
368     memcpy (nue->info, d, cantBytes); CRLE
369     nue->tamInfo = cantBytes; CRLE
370     nue->sig = *p; CRLE
371     *p = nue; CRLE
372     return 1; CRLE
373 } CRLE
374 CRLE
375 int sacarPrimerLista (tLista *p, void *d, unsigned cantBytes) CRLE
376 { CRLE
377     tNodo *aux = *p; CRLE
378     CRLE
379     if (aux == NULL) CRLE
380         return 0; CRLE
381     *p = aux->sig; CRLE
382     memcpy (d, aux->info, minimo (cantBytes, aux->tamInfo)); CRLE
383     free (aux->info); CRLE
384     free (aux); CRLE
385     return 1; CRLE
386 } CRLE
387 CRLE
388 int verPrimerLista (const tLista *p, void *d, unsigned cantBytes) CRLE
389 { CRLE
390     if (*p == NULL) CRLE
391         return 0; CRLE
392     memcpy (d, (*p)->info, minimo (cantBytes, (*p)->tamInfo)); CRLE
393     return 1; CRLE
394 } CRLE
395 CRLE
396 int ponerAlFinal (tLista *p, const void *d, unsigned cantBytes) CRLE
397 { CRLE
398     tNodo *nue; CRLE
399     CRLE
400     while (*p) CRLE
401         *p = &(*p)->sig; CRLE
402     if ((nue = (tNodo *) malloc (sizeof (tNodo))) == NULL || CRLE
403         (nue->info = malloc (cantBytes)) == NULL) CRLE
404     { CRLE
405         free (nue); CRLE
406         return 0; CRLE
407     } CRLE
408     memcpy (nue->info, d, cantBytes); CRLE
409     nue->tamInfo = cantBytes; CRLE
410     nue->sig = NULL; CRLE
411     *p = nue; CRLE
412     return 1; CRLE
413 } CRLE
414 CRLE

```

```

415 int sacarUltimoLista(tLista *p, void *d, unsigned cantBytes) CRLE
416 { CRLE
417     if (*p == NULL) CRLE
418         return 0; CRLE
419     while ((*p) -> sig) CRLE
420         p = &(*p) -> sig; CRLE
421     memcpy(d, (*p) -> info, minimo(cantBytes, (*p) -> tamInfo)); CRLE
422     free((*p) -> info); CRLE
423     free(*p); CRLE
424     *p = NULL; CRLE
425     return 1; CRLE
426 } CRLE
427 CRLE
428 int verUltimoLista(const tLista *p, void *d, unsigned cantBytes) CRLE
429 { CRLE
430     if (*p == NULL) CRLE
431         return 0; CRLE
432     while ((*p) -> sig) CRLE
433         p = &(*p) -> sig; CRLE
434     memcpy(d, (*p) -> info, minimo(cantBytes, (*p) -> tamInfo)); CRLE
435     return 1; CRLE
436 } CRLE
437 CRLE
438 int mostrarLista(const tLista *p, void (*mostrar)(const void *)) CRLE
439 { CRLE
440     int cant = 0; CRLE
441     CRLE
442     if (*p) CRLE
443         mostrar(NULL); CRLE
444     while (*p) CRLE
445     { CRLE
446         mostrar((*p) -> info); CRLE
447         p = &(*p) -> sig; CRLE
448         cant++; CRLE
449     } CRLE
450     return cant; CRLE
451 } CRLE
452 CRLE
453 int mostrarListaAlReves(const tLista *p, void (*mostrar)(const void *)) CRLE
454 { CRLE
455     if (*p) CRLE
456     { CRLE
457         int cant = mostrarListaAlReves(&(*p) -> sig, mostrar); CRLE
458         mostrar((*p) -> info); CRLE
459         return cant + 1; CRLE
460     } CRLE
461     mostrar(NULL); CRLE
462     return 0; CRLE
463 } CRLE
464 CRLE
465 void ordenarLista(tLista *p, int (*comparar)(const void *, const void *)) CRLE
466 { CRLE
467     int marca = 1; CRLE
468     CRLE
469     if (*p == NULL) CRLE
470         return; CRLE
471     while (marca) CRLE
472     { CRLE
473         marca = 0; CRLE
474         tLista *q = p; CRLE
475         while ((*q) -> sig) CRLE
476             { CRLE
477                 if (comparar((*q) -> info, (*q) -> sig -> info) > 0) CRLE
478                 { CRLE
479                     void *infoAux = (*q) -> info; CRLE
480                     unsigned tamAux = (*q) -> tamInfo; CRLE
481                     (*q) -> info = (*q) -> sig -> info; CRLE
482                     (*q) -> sig -> info = infoAux; CRLE
483                     (*q) -> tamInfo = (*q) -> sig -> tamInfo; CRLE

```

```

484 ..... (*q)->sig->tamInfo = tamAux; CRLE
485 ..... marca = 1; CRLE
486 ..... } CRLE
487 ..... q = &(*q)->sig; CRLE
488 ..... } CRLE
489 ..... } CRLE
490 } CRLE
491 CRLE
492 int ponerEnOrdenOAcumular(tLista *p, const void *d, unsigned cantBytes, CRLE
493 ..... int (*comparar) (const void *, const void *), CRLE
494 ..... void (*acumular) (void **, const void *, CRLE
495 ..... unsigned *, unsigned)) CRLE
496 { CRLE
497 ..... tNodo *nue; CRLE
498 CRLE
499 ..... while (*p && comparar(d, (*p)->info) > 0) CRLE
500 ..... p = &(*p)->sig; CRLE
501 ..... if (*p && comparar(d, (*p)->info) == 0) CRLE
502 ..... { CRLE
503 ..... if (acumular) CRLE
504 ..... acumular(&(*p)->info, d, &(*p)->tamInfo, cantBytes); CRLE
505 ..... return CLA_DUP; CRLE
506 ..... } CRLE
507 ..... if ((nue = (tNodo *) malloc(sizeof(tNodo))) == NULL || CRLE
508 ..... (nue->info = malloc(cantBytes)) == NULL) CRLE
509 ..... { CRLE
510 ..... free(nue); CRLE
511 ..... return SIN_MEM; CRLE
512 ..... } CRLE
513 ..... memcpy(nue->info, d, cantBytes); CRLE
514 ..... nue->tamInfo = cantBytes; CRLE
515 ..... nue->sig = *p; CRLE
516 ..... *p = nue; CRLE
517 ..... return TODO_BIEN; CRLE
518 } CRLE
519 CRLE
520

```