```c
/* --------------------o---x---o--------------------
 *           main.h
 * --------------------o---x---o-------------------- */
#ifndef MAIN_H_
#define MAIN_H_

#include <stdio.h>

#include "productos/productos.h"
#include "lista/lista.h"


void probarIngresarYMostrarProd(void);

void probarPonerAdelanteVerUltimoVerPrimeroYVaciar(void);

void probarPonerAtrasSacarUltimoSacarPrimeroYMostrar(void);

void probarPonerEnOrdenMostrarAlRevesYOrdenar(void);


#endif
/* --------------------o---x---o--------------------
 *           main.c
 * --------------------o---x---o-------------------- */

#include "main.h"


int main(void)
{
    probarIngresarYMostrarProd();

    probarPonerAdelanteVerUltimoVerPrimeroYVaciar();

    probarPonerAtrasSacarUltimoSacarPrimeroYMostrar();

    probarPonerEnOrdenMostrarAlRevesYOrdenar();

    return 0;
}


void probarIngresarYMostrarProd(void)
{
    tProd    prod;
    int      result,
             cant = 0;

    puts("\nProbando ingresar productos y mostrar productos");
    result = ingresarProducto(&prod);
    if(result)
        mostrarProducto(NULL);
    while(result)
    {
        mostrarProducto(&prod);
        result = ingresarProducto(&prod);
        cant++;
    }
    fprintf(stdout, "Se mostraron %d productos.\n", cant);
}

void probarPonerAdelanteVerUltimoVerPrimeroYVaciar(void)
{
    tLista      lista;
    tProd       prod;
    int         cant = 0;

    puts("\nFUNCION: probarPonerAdelanteVerUltimoVerPrimeroYVaciar");
```

```
    crearLista(&lista);
}

    printf("Insertando al comienzo de la lista\n");
    while(ingresarProducto(&prod) &&
          ponerAlComienzo(&lista, &prod, sizeof(tProd)))
    {
        if(cant == 0)
            mostrarProducto(NULL);
        mostrarProducto(&prod);
        cant++;
    }
    printf("Se insertaron %d productos en la lista\n", cant);
    if(verUltimoLista(&lista, &prod, sizeof(prod)))
    {
        printf("Ultimo de la lista\n");
        mostrarProducto(NULL);
        mostrarProducto(&prod);
    }
    if(verPrimeroLista(&lista, &prod, sizeof(prod)))
    {
        printf("Primero de la lista\n");
        mostrarProducto(NULL);
        mostrarProducto(&prod);
    }
    vaciarLista(&lista);
}

void probarPonerAtrasSacarUltimoSacarPrimeroYMostrar(void)
{
    tLista      lista;
    tProd       prod;
    int         cant = 0;

    puts("\nFUNCION: probarPonerAtrasSacarUltimoSacarPrimeroYMostrar");
    crearLista(&lista);

    printf("Insertando al final de la lista\n");
    while(ingresarProducto(&prod) &&
          ponerAlFinal(&lista, &prod, sizeof(tProd)))
    {
        if(cant == 0)
            mostrarProducto(NULL);
        mostrarProducto(&prod);
        cant++;
    }
    printf("Se insertaron %d productos en la lista\n", cant);
    if(sacarPrimeroLista(&lista, &prod, sizeof(prod)))
    {
        printf("Primero de la lista\n");
        mostrarProducto(NULL);
        mostrarProducto(&prod);
    }
    if(sacarUltimoLista(&lista, &prod, sizeof(prod)))
    {
        printf("Ultimo de la lista\n");
        mostrarProducto(NULL);
        mostrarProducto(&prod);
    }
    cant = mostrarLista(&lista, mostrarProducto);
    if(cant)
        printf("Se mostraron %d productos\n", cant);
    vaciarLista(&lista);
}

void probarPonerEnOrdenMostrarAlRevesYOrdenar(void)
{
    tLista      lista;
    tProd       prod;
```

```
    int cant = 0,
        dupl = 0;

    puts("\nFUNCION: probarPonerEnOrdenMostrarAlRevesYOrdenar");
    crearLista(&lista);

    printf("Insertando en orden por clave del producto\n");
    while(ingresarProducto(&prod))

    {
        int result = ponerEnOrdenOAcumular(&lista, &prod, sizeof(tProd),
                                           compararProdXCodProd, NULL);
        if(cant == 0)
            mostrarProducto(NULL);
        mostrarProducto(&prod);
        if(result == TODO_BIEN)
            cant++;
        if(result == CLA_DUP)
            dupl++;
    }
    printf("Se insertaron %d productos en la lista y hubo %d duplicados\n",
           cant, dupl);
    cant = mostrarListaAlReves(&lista, mostrarProducto);
    if(cant)
        printf("Se mostraron %d productos\n", cant);
    puts("Ordenando por descripcion del producto");
    ordenarLista(&lista, compararProdXDescrip);
    cant = mostrarListaAlReves(&lista, mostrarProducto);
    if(cant)
        printf("Se mostraron %d productos\n", cant);
    vaciarLista(&lista);
}

/* ---------------------o---x---o---------------------
 *          productos.h
 * ---------------------o---x---o--------------------- */
#ifndef PRODUCTOS_H_
#define PRODUCTOS_H_

#include <stdio.h>
#include <string.h>


typedef struct
{
    char    codProd[11],
            descrip[46];
} tProd;

int ingresarProducto(tProd *d);

void mostrarProducto(const void *d);

int compararProdXCodProd(const void *d1, const void *d2);

int compararProdXDescrip(const void *d1, const void *d2);


#endif
/* ---------------------o---x---o---------------------
 *          productos.c
 * ---------------------o---x---o--------------------- */

#include "productos.h"


int ingresarProducto(tProd *d)
{
    static const tProd productos[] = {
```

```
        ///1234567890    123456789 123456789 123456789 123456789 12345
        { "clavoro3/4", "Clavo de oro 24 kilates de 3/4 de pulgada" },
        { "martillo3K", "Martillo bolita con saca clavos de 3 kilos"},
        { "yesoalam1", "Alambre de yeso de un milimetro de espesor" },
        { "vidrem-15", "Remache de vidrio de 1,5 milimetros" },
        { "plom-telgo", "Plomada de poliestireno expandido" },
        { "limagoma17", "Lima de goma de 17 pulgadas"} };
    static int posi = 0;

    if(posi == sizeof(productos) / sizeof(tProd))
    {
        posi = 0;
        return 0;
    }
    *d = productos[posi];
    posi++;

    return 1;
}

void mostrarProducto(const void *d)
{
    tProd *dProd = (tProd *)d;
    if(d)
        fprintf(stdout,
                "%-*s %-*s ...\n",
                sizeof(dProd->codProd) - 1, dProd->codProd,
                sizeof(dProd->descrip) - 1, dProd->descrip);
    else
        fprintf(stdout,
                "%-*.*s %-*.*s ...\n",
                sizeof(dProd->codProd) - 1, sizeof(dProd->codProd) - 1,
                    "Cod. Producto",
                sizeof(dProd->descrip) - 1, sizeof(dProd->descrip) - 1,
                    "Descripcion del producto");
}

int compararProdXCodProd(const void *d1, const void *d2)
{
    tProd *p = (tProd *)d1,
          *q = (tProd *)d2;

    return strcmpi(p->codProd, q->codProd);
}

int compararProdXDescrip(const void *d1, const void *d2)
{
    tProd *p = (tProd *)d1,
          *q = (tProd *)d2;

    return strcmpi(p->descrip, q->descrip);
}

/* --------------------o---x---o--------------------
 *          lista.h
 * --------------------o---x---o-------------------- */
#ifndef LISTA_H_
#define LISTA_H_

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

#define SIN_MEM         1
#define CLA_DUP         2
#define TODO_BIEN       0

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

typedef struct sNodo
```

```
{CRLF
    void           *info;CRLF
    unsigned       tamInfo;CRLF
    struct sNodo   *sig;CRLF
} tNodo;CRLF
typedef tNodo *tLista;CRLF
CRLF
CRLF
void crearLista(tLista *p);CRLF
CRLF
int  listaVacia(const tLista *p);CRLF
CRLF
int  listaLlena(const tLista *p, unsigned cantBytes);CRLF
CRLF
void vaciarLista(tLista *p);CRLF
CRLF
int  ponerAlComienzo(tLista *p, const void *d, unsigned cantBytes);CRLF
CRLF
CRLF
int  sacarPrimeroLista(tLista *p, void *d, unsigned cantBytes);CRLF
CRLF
int  verPrimeroLista(const tLista *p, void *d, unsigned cantBytes);CRLF
CRLF
int  ponerAlFinal(tLista *p, const void *d, unsigned cantBytes);CRLF
CRLF
int  sacarUltimoLista(tLista *p, void *d, unsigned cantBytes);CRLF
CRLF
int  verUltimoLista(const tLista *p, void *d, unsigned cantBytes);CRLF
CRLF
CRLF
int  mostrarLista(const tLista *p, void (*mostrar)(const void *));CRLF
CRLF
int  mostrarListaAlReves(const tLista *p, void (*mostrar)(const void *));CRLF
CRLF
void ordenarLista(tLista *p, int (*comparar)(const void *, const void *));CRLF
CRLF
int  ponerEnOrdenOAcumular(tLista *p, const void *d, unsigned cantBytes,CRLF
                           int (*comparar)(const void *, const void *),CRLF
                           void (*acumular)(void **, const void *,CRLF
                                            unsigned *, unsigned));CRLF
CRLF
CRLF
#endifCRLF
/* ---------------------o---x---o---------------------CRLF
 *          lista.cCRLF
 * ---------------------o---x---o--------------------- */CRLF
CRLF
#include "lista.h"CRLF
CRLF
void crearLista(tLista *p)CRLF
{CRLF
    *p = NULL;CRLF
}CRLF
CRLF
int  listaVacia(const tLista *p)CRLF
{CRLF
    return *p == NULL;CRLF
}CRLF
CRLF
int  listaLlena(const tLista *p, unsigned cantBytes)CRLF
{CRLF
    tNodo  *aux = (tNodo *)malloc(sizeof(tNodo));CRLF
    void   *info = malloc(cantBytes);CRLF
CRLF
    free(aux);CRLF
    free(info);CRLF
    return aux == NULL || info == NULL;CRLF
}CRLF
CRLF
```

```
void vaciarLista(tLista *p)CRLF
{CRLF
    while(*p)CRLF
    {CRLF
        tNodo *aux = *p;CRLF
CRLF
        *p = aux->sig;CRLF
        free(aux->info);CRLF
        free(aux);CRLF
    }CRLF
}CRLF
CRLF
int ponerAlComienzo(tLista *p, const void *d, unsigned cantBytes)CRLF
{CRLF
    tNodo *nue;CRLF
CRLF
    if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||CRLF
       (nue->info = malloc(cantBytes)) == NULL)CRLF
    {CRLF
        free(nue);CRLF
        return 0;CRLF
    }CRLF
    memcpy(nue->info, d, cantBytes);CRLF
    nue->tamInfo = cantBytes;CRLF
    nue->sig = *p;CRLF
    *p = nue;CRLF
    return 1;CRLF
}CRLF
CRLF
int sacarPrimeroLista(tLista *p, void *d, unsigned cantBytes)CRLF
{CRLF
    tNodo *aux = *p;CRLF
CRLF
    if(aux == NULL)CRLF
        return 0;CRLF
    *p = aux->sig;CRLF
    memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));CRLF
    free(aux->info);CRLF
    free(aux);CRLF
    return 1;CRLF
}CRLF
CRLF
int verPrimeroLista(const tLista *p, void *d, unsigned cantBytes)CRLF
{CRLF
    if(*p == NULL)CRLF
        return 0;CRLF
    memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));CRLF
    return 1;CRLF
}CRLF
CRLF
int ponerAlFinal(tLista *p, const void *d, unsigned cantBytes)CRLF
{CRLF
    tNodo *nue;CRLF
CRLF
    while(*p)CRLF
        p = &(*p)->sig;CRLF
    if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||CRLF
       (nue->info = malloc(cantBytes)) == NULL)CRLF
    {CRLF
        free(nue);CRLF
        return 0;CRLF
    }CRLF
    memcpy(nue->info, d, cantBytes);CRLF
    nue->tamInfo = cantBytes;CRLF
    nue->sig = NULL;CRLF
    *p = nue;CRLF
    return 1;CRLF
}CRLF
CRLF
```

```c
int sacarUltimoLista(tLista *p, void *d, unsigned cantBytes)
{
    if(*p == NULL)
        return 0;
    while((*p)->sig)
        p = &(*p)->sig;
    memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
    free((*p)->info);
    free(*p);
    *p = NULL;
    return 1;
}

int verUltimoLista(const tLista *p, void *d, unsigned cantBytes)
{
    if(*p == NULL)
        return 0;
    while((*p)->sig)
        p = &(*p)->sig;
    memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
    return 1;
}

int mostrarLista(const tLista *p, void (*mostrar)(const void *))
{
    int cant = 0;

    if(*p)
        mostrar(NULL);
    while(*p)
    {
        mostrar((*p)->info);
        p = &(*p)->sig;
        cant++;
    }
    return cant;
}

int mostrarListaAlReves(const tLista *p, void (*mostrar)(const void *))
{
    if(*p)
    {
        int cant = mostrarListaAlReves(&(*p)->sig, mostrar);
        mostrar((*p)->info);
        return cant + 1;
    }
    mostrar(NULL);
    return 0;
}

void ordenarLista(tLista *p, int (*comparar)(const void *, const void *))
{
    int marca = 1;

    if(*p == NULL)
        return;
    while(marca)
    {
        marca = 0;
        tLista *q = p;
        while((*q)->sig)
        {
            if(comparar((*q)->info, (*q)->sig->info) > 0)
            {
                void *infoAux = (*q)->info;
                unsigned tamAux = (*q)->tamInfo;
                (*q)->info = (*q)->sig->info;
                (*q)->sig->info = infoAux;
                (*q)->tamInfo = (*q)->sig->tamInfo;
```

```
                    (*q)->sig->tamInfo = tamAux;CRLF
                    marca = 1;CRLF
                }CRLF
                q = &(*q)->sig;CRLF
            }CRLF
        }CRLF
    }CRLF
    CRLF
    int ponerEnOrdenOAcumular(tLista *p, const void *d, unsigned cantBytes,CRLF
                              int (*comparar)(const void *, const void *),CRLF
                              void (*acumular)(void **, const void *,CRLF
                                               unsigned *, unsigned))CRLF
    {CRLF
        tNodo *nue;CRLF
    CRLF
        while(*p && comparar(d, (*p)->info) > 0)CRLF
            p = &(*p)->sig;CRLF
        if(*p && comparar(d, (*p)->info) == 0)CRLF
        {CRLF
            if(acumular)CRLF
                acumular(&(*p)->info, d, &(*p)->tamInfo, cantBytes);CRLF
            return CLA_DUP;CRLF
        }CRLF
        if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||CRLF
           (nue->info = malloc(cantBytes)) == NULL)CRLF
        {CRLF
            free(nue);CRLF
            return SIN_MEM;CRLF
        }CRLF
        memcpy(nue->info, d, cantBytes);CRLF
        nue->tamInfo = cantBytes;CRLF
        nue->sig = *p;CRLF
        *p = nue;CRLF
        return TODO_BIEN;CRLF
    }CRLF
    CRLF
```