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

#include <stdio.h>


#include "productos/productos.h"
#include "colaEstatica/cola.h"
#include "colaDinamica/cola.h"


void probarIngresarYMostrarProd(void);

void probarPonerYSacarDeCola(void);

#endif
/* ----------------------o---x---o----------------------
 *          main.c
 * ----------------------o---x---o---------------------- */
#include "main.h"


int main(void)
{
    printf("%d %d\n", sizeof(unsigned), sizeof(tProd));
    probarIngresarYMostrarProd();

    probarPonerYSacarDeCola();

    return 0;
}


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

    puts("Probando 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\n", cant);
}


void probarPonerYSacarDeCola(void)
{
    tProd   prod;
    tCola   cola;
    int     result,
            llena;

    crearCola(&cola);
    llena = colaLlena(&cola, sizeof(tProd));
    if(!llena)
    {
        result = ingresarProducto(&prod);
        puts("Procediendo a poner en cola");
        mostrarProducto(NULL);
    }
    while(result && !llena)
    {

        if(!ponerEnCola(&cola, &prod, sizeof(tProd)))
        {
            fprintf(stderr, "ERROR - inesperado: cola llena\n");
            puts("no se pudo cargar la informacion");
        }
        mostrarProducto(&prod);
        llena = colaLlena(&cola, sizeof(tProd));
        if(!llena)
            result = ingresarProducto(&prod);
        else
            puts("Se lleno la cola");
    }

    puts("\nMostrando el primero de la cola");
    if(!colaVacia(&cola))
    {
        tProd otro;
        verPrimeroCola(&cola, &otro, sizeof(tProd));
        mostrarProducto(&otro);
    }
    else
        puts("La cola estaba vacia");

    puts("\nProcediendo a sacar de la cola y mostrar");
    if(colaVacia(&cola))
        puts("La cola está vacía");
    else
        mostrarProducto(NULL);
    while(sacarDeCola(&cola, &prod, sizeof(tProd)))
        mostrarProducto(&prod);
    puts("");
}

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

#include <stdio.h>


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

int ingresarProducto(tProd *d);

void mostrarProducto(const tProd *d);


#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"},
        { "alamyeso1",  "Alambre de yeso de un milimetro de espesor" },
        { "rem-vid15",  "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 tProd *d)
{
    if(d)
        fprintf(stdout,
                "%-*s %-*s ...\n",
                sizeof(d->codProd) - 1, d->codProd,
                sizeof(d->descrip) - 1, d->descrip);
    else
        fprintf(stdout,
                "%-*.*s %-*.*s ...\n",
                sizeof(d->codProd) - 1, sizeof(d->codProd) - 1,
                    "Cod. Producto",
                sizeof(d->descrip) - 1, sizeof(d->descrip) - 1,
                    "Descripcion del producto");
}

/* ---------------------o---x---o---------------------
 *          cola.h     ESTÁTICA
 * ---------------------o---x---o--------------------- */
#ifdef ESTATICA

#ifndef COLA_H_
#define COLA_H_


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

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

#define     TAM_COLA        300

typedef struct
{
    char        cola[TAM_COLA];
    unsigned    pri,
                ult,
                tamDisp;
} tCola;


void crearCola(tCola *p);
int  colaLlena(const tCola *p, unsigned cantBytes);
int  ponerEnCola(tCola *p, const void *d, unsigned cantBytes);
int  verPrimeroCola(const tCola *p, void *d, unsigned cantBytes);
int  colaVacia(const tCola *p);
int  sacarDeCola(tCola *p, void *d, unsigned cantBytes);
void vaciarCola(tCola *p);

#endif
```

```c

#endif
/* ---------------------o---x---o---------------------
 *          cola.c      ESTÁTICA
 * ---------------------o---x---o--------------------- */
#ifdef ESTATICA


#include "cola.h"


void crearCola(tCola *p)
{
    p->pri      = TAM_COLA - 70;
    p->ult      = TAM_COLA - 70;
    p->tamDisp  = TAM_COLA;
}

int  colaLlena(const tCola *p, unsigned cantBytes)
{
    return p->tamDisp < cantBytes + sizeof(unsigned);
}

int  ponerEnCola(tCola *p, const void *d, unsigned cantBytes)
{
    unsigned    ini,
                fin;

    if(p->tamDisp < sizeof(unsigned) + cantBytes)
        return 0;
    p->tamDisp -= sizeof(unsigned) + cantBytes;
    if((ini = minimo(sizeof(cantBytes), TAM_COLA - p->ult)) != 0)
        memcpy(p->cola + p->ult, &cantBytes, ini);
    if((fin = sizeof(cantBytes) - ini) != 0)
        memcpy(p->cola, ((char *)&cantBytes) + ini, fin);
    p->ult = fin ? fin : p->ult + ini;
    if((ini = minimo(cantBytes, TAM_COLA - p->ult)) != 0)
        memcpy(p->cola + p->ult, d, ini);
    if((fin = cantBytes - ini) != 0)
        memcpy(p->cola, ((char *)d) + ini, fin);
    p->ult = fin ? fin : p->ult + ini;
    return 1;
}

int  verPrimeroCola(const tCola *p, void *d, unsigned cantBytes)
{
    unsigned    tamInfo,
                ini,
                fin,
                pos = p->pri;

    if(p->tamDisp == TAM_COLA)
        return 0;
    if((ini = minimo(sizeof(unsigned), TAM_COLA - pos)) !=0)
        memcpy(&tamInfo, p->cola + pos, ini);
    if((fin = sizeof(unsigned) - ini) != 0)
        memcpy(((char *)&tamInfo) + ini, p->cola, fin);
    pos = fin ? fin : pos + ini;
    tamInfo = minimo(tamInfo, cantBytes);
    if((ini = minimo(tamInfo, TAM_COLA - pos)) != 0)
        memcpy(d, p->cola + pos, ini);
    if((fin = tamInfo - ini) != 0)
        memcpy(((char *)d) + ini, p->cola, fin);
    return 1;
}

int  colaVacia(const tCola *p)
{
    return p->tamDisp == TAM_COLA;
}

int  sacarDeCola(tCola *p, void *d, unsigned cantBytes)
{
    unsigned    tamInfo,
                ini,
                fin;

    if(p->tamDisp == TAM_COLA)
        return 0;
    if((ini = minimo(sizeof(unsigned), TAM_COLA - p->pri)) != 0)
        memcpy(&tamInfo, p->cola + p->pri, ini);
    if((fin = sizeof(unsigned) - ini) != 0)
        memcpy(((char *)&tamInfo) + ini, p->cola, fin);
    p->pri = fin ? fin : p->pri + ini;
    tamInfo = minimo(tamInfo, cantBytes);
    p->tamDisp += sizeof(unsigned) + tamInfo;
    if((ini = minimo(tamInfo, TAM_COLA - p->pri)) != 0)
        memcpy(d, p->cola + p->pri, ini);
    if((fin = tamInfo - ini) != 0)
        memcpy(((char *)d) + ini, p->cola, fin);
    p->pri = fin ? fin : p->pri + ini;
    return 1;
}

void vaciarCola(tCola *p)
{
    p->ult = p->pri;
    p->tamDisp = TAM_COLA;
}

#endif

/* ---------------------o---x---o---------------------
 *          cola.h      DINÁMICA
 * ---------------------o---x---o--------------------- */
#ifdef DINAMICA

#ifndef COLA_H_
#define COLA_H_

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

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

typedef struct sNodo
{
    void           *info;
    unsigned        tamInfo;
    struct sNodo   *sig;
} tNodo;

typedef struct
{
    tNodo  *pri,
           *ult;
} tCola;

void crearCola(tCola *p);
int  colaLlena(const tCola *p, unsigned cantBytes);
int  ponerEnCola(tCola *p, const void *d, unsigned cantBytes);
int  verPrimeroCola(const tCola *p, void *d, unsigned cantBytes);
int  colaVacia(const tCola *p);
int  sacarDeCola(tCola *p, void *d, unsigned cantBytes);
void vaciarCola(tCola *p);

#endif

```

```c
#endif
/* ---------------------o---x---o---------------------
 *          cola.c     DINÁMICA
 * ---------------------o---x---o--------------------- */
#ifdef DINAMICA

#include "cola.h"


void crearCola(tCola *p)
{
    p->pri = NULL;
    p->ult = NULL;
}

int  colaLlena(const tCola *p, unsigned cantBytes)
{
    tNodo  *aux = (tNodo *)malloc(sizeof(tNodo));
    void   *info = malloc(cantBytes);
    free(aux);
    free(info);
    return aux == NULL || info == NULL;
}

int  ponerEnCola(tCola *p, const void *d, unsigned cantBytes)
{
    tNodo *nue = (tNodo *) malloc(sizeof(tNodo));

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

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

int  colaVacia(const tCola *p)
{
    return p->pri == NULL;
}

int  sacarDeCola(tCola *p, void *d, unsigned cantBytes)
{
    tNodo *aux = p->pri;
    if(aux == NULL)
        return 0;
    p->pri = aux->sig;
    memcpy(d, aux->info, minimo(aux->tamInfo, cantBytes));
    free(aux->info);
    free(aux);
    if(p->pri == NULL)
        p->ult = NULL;
    return 1;
}

void vaciarCola(tCola *p)
{
    while(p->pri)
    {
        tNodo *aux = p->pri;
        p->pri = aux->sig;
        free(aux->info);
        free(aux);
    }
    p->ult = NULL;
}

#endif

```