```
/**//* 27.940.642-ROSSI,SebastianPablo-(05-1965) *//**/

/**
        Ejercicio TDA: Rehacer
        Observaciones: La función mostrarMovim_MIO no funciona correctamente.
        La funcioón mostrarTotal no calcula los importes correctamente. La función
        ordenarLista_MIO posee una condicion en la que hay tres ciclos while anidados.
        Revisar el algoritmo de ordenamiento.
**/

/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
/**//**//* CUALQUIER INCLUDE DE BIBLIOTECA QUE NECESITE, HÁGALO ACÁ   *//**//**/


/**//**//* CUALQUIER INCLUDE DE BIBLIOTECA QUE NECESITE, HÁGALO ACÁ   *//**//**/
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
#include "funciones.h"
#include <string.h>
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
/**//**//* ACÁ DEBE DESARROLLAR LAS FUNCIONES Y PRIMITIVAS PEDIDAS    *//**//**/
/**//**//* ADEMÁS DE CUALQUIER OTRA FUNCIÓN QUE SE REQUIERA           *//**//**/



/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
/**//* FUNCIONES Y PRIMITIVAS A DESARROLLAR                                  *//**/
/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
/**//* para la información                                                   *//**/

void mostrarMovim_MIO(const void *d, FILE *fp)
{
    if(d)
        fprintf(fp,"%-*s %8.2lf\n",sizeof(((tMovi *)d)->ctaCte), ((tMovi *)d)->ctaCte, ((tMovi
*)d)->saldo);
    else
        fprintf(fp,"Nro Cuenta Banc   Importe\n");
}

int compararMovim_MIO(const void *d1, const void *d2)
{
    char *daux1 = ((tMovi *)d1)->ctaCte;
    char *daux2 = ((tMovi *)d2)->ctaCte;

    while(*daux1 && *daux2)
    {
        if(*daux1!=*daux2)
            return *daux1-*daux2;
        daux1++;
        daux2++;
    }
    return 0;
}


int esCtaCte002_MIO(const void *d)
{
    return !strcmp(strrchr(((tMovi *)d)->ctaCte,'-')+1,"002");
}


int acumularMoviSaldo_MIO(void **dest, unsigned *tamDest,
                      const void *ori, unsigned tamOri)
{
    ((tMovi *)*dest)->saldo += ((tMovi *)ori)->saldo;
    return 1;
}


void mostrarTotal_MIO(const void *d, FILE *fp)
```

```
{
     if(fp)
        fprintf(fp, "%15s %9.2lf\n\n", "Total cliente:",((tMovi *)d)->saldo);
}


/**//* para el TDA LISTA                                                    *//**/

int mostrarLista_MIO(const tLista *p,
                     void (*mostrar)(const void *, FILE *), FILE *fp)
{
    int eslabones = 0;
    mostrar(NULL, fp);
    while(*p)
    {
        mostrar((*p)->info, fp);
        p = &(*p)->sig;
        eslabones++;
    }
    return eslabones;
}


void ordenarLista_MIO(tLista *p, int (*comparar)(const void *, const void *))
{
    tNodo *aux, *minimo = NULL;
    tNodo *inicio, *comparador, *menor_actual;

    inicio = *p;
    comparador = inicio->sig;
    menor_actual = inicio;


    while(inicio)
    {
        // mientras este en la lista comparara si
        // hay un eslabon con info menor a menor_actual
        while(comparador)
        {
            if(comparar(comparador->info, menor_actual->info)<0)
            {
                // si encuentra un dato menor
                // corre a menor_actual hasta la posicion de comparador
                while(menor_actual != comparador)
                {
                    // guarda el anterior a menor_actual para rearmar la lista
                    aux = menor_actual;
                    menor_actual = menor_actual->sig;
                }
            }
            // continua moviendo el comparador para ver si hay otro que sea
            // menor al actual
            comparador = comparador->sig;
        }

        // si el inicio de comparacion es el minimo
        // y/o es el primero de la lista
        // el siguiente pasa a ser el proximo a ser comienzo de donde comparar
        if(inicio == menor_actual)
        {
            minimo = inicio;
            inicio = inicio->sig;
        }
        else
        {
            // si es el menor de toda la lista
            // pasa a ser el primero
            if(!minimo)
            {
```

```
                minimo = menor_actual;
                aux->sig = menor_actual->sig;
                minimo->sig = *p;
                *p = minimo;
            }
            // sino, se mueve el ultimo ingresado y el proximo menor
            // se ingresara despues de este
            else
            {
                    minimo->sig = menor_actual;
                    minimo = menor_actual;
                    aux->sig = menor_actual->sig;
                    minimo->sig = inicio;
            }
        }

        // se retorna el menor_actual al minimo no cambiado
        menor_actual = inicio;
        if(inicio)
            comparador = menor_actual->sig;
    }

}


int eliminarMostrarYMostrarSubTot_MIO(tLista *p, FILE *fpPant,
                                      int comparar(const void *, const void *),
                                      int comparar2(const void *d),
                                      int acumular(void **, unsigned *,
                                                   const void *, unsigned),
                                      void mostrar(const void *, FILE *),
                                      void mostrar2(const void *, FILE *))
{
    tNodo *aux;
    int eslabones = 0;

    while(*p)
    {
        if(comparar2((*p)->info))
        {
            mostrar(NULL, fpPant);
            mostrar((*p)->info, fpPant);

            while((*p)->sig &&  !comparar((*p)->info, (*p)->sig->info) )
            {
                acumular(&(*p)->info,&(*p)->tamInfo,(*p)->sig->info,(*p)->sig->tamInfo);

                mostrar((*p)->sig->info, fpPant);

                aux = (*p)->sig;
                (*p)->sig = aux->sig;

                free(aux->info);
                free(aux);
                eslabones++;
            }
            mostrar2((*p)->info, fpPant);
            aux = *p;
            *p = aux->sig;

            eslabones++;
            free(aux->info);
            free(aux);
        }
        else
            p = &(*p)->sig;
     }
     return eslabones;
}
```

```
int  vaciarListaYMostrar_MIO(tLista *p,
                             void (*mostrar)(const void *, FILE *), FILE *fp)
{
    tNodo *aux;
    size_t nodos = 0;
    while(*p)
    {
        aux=*p;
        *p=aux->sig;
        if(mostrar && fp)
            mostrar(aux->info, fp);
        free(aux->info);
        free(aux);
        nodos++;
    }
    return nodos;
}

/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
```