PARA COMPILAR

```
gcc DatosPersonalesB1.c comunes.c principal.c -o compara.exe
gcc DatosPersonalesB2.c comunes.c principal.c -o compara.exe
gcc DatosPersonalesT1.c comunes.c principal.c -o compara.exe
gcc DatosPersonalesT2.c comunes.c principal.c -o compara.exe
gcc comunes.c borradoFisico.c DatosPersonalesB2.c –o borradoFisico.exe
```

ESTUCTURA DE LOS DATOS EN MEMORIA

```
struct DatosPersonales
{
  long dni; //clave unica para cada registro
  char nombre[15];
  char apellido[15];
  float salario;
};
```

#DEFINE

```
#define MAX_LINEA 256
#define NUM_CAMPOS 4
#define BORRADO -1
```

T1

ESTUCTURA DE LOS DATOS EN DISCO: T1

Presupone que la información de cada persona está en una única línea.

NO se permiten nombres ni apellidos compuestos (con espacios en blanco)

ej.

```
1111 antonio gomez 2231.300
2222 francisco perez 2345.500
3333 lucia lozano 2111.420
```

T2

ESTRUCTURA DE LOS DATOS EN DISCO: T2

Presupone que la información de cada persona ocupa tres líneas, una por campo.

SI se permiten nombres y apellidos compuestos (con espacios en blanco)

ej.

```
1111
antonio jose
gomez
1234.220
2222
francisco
del monte
1345.650
3333
lucia
lozano
956.800
```

B1 y B2

ESTRUCTURA DE LOS DATOS EN DISCO: B1 y B2

```
struct DatosPersonales
{
  char nombre[15];
  char apellido[15];
  long dni; //clave única para cada registro
  float salario;
  };
```

NOTAS:

- No es obligatorio utilizar una marca de borrado, eso dependerá del enunciado del problema.
- En el caso B1 suponemos que todos los DNI son válidos (no hay borrados)
- En el caso B2 la marca de borrado consistirá en poner el campo dni a -1

void verFichero(char *fichero)

```
void verFichero(char *nombreFichero)
{
 FILE *pFichero;
 struct DatosPersonales persona;
 /* abre fichero para lectura */
 pFichero = fopen(nombreFichero, "r");
```

```
void verFichero(char* nombreFichero)
{
 FILE *pFichero;
 struct DatosPersonales persona;
 char linea[MAX_LINEA];
 /* abre fichero para lectura */
 pFichero = fopen(nombreFichero, "r");
```

```
int borrarporDni(char *nombreFichero,
long dni_buscar)
{
 FILE *pFichero1, *pFichero2;
 struct DatosPersonales persona;
 char linea[MAX_LINEA];
 int borrado = 0;
```

```
void verFichero(char *nombreFichero)
{
 FILE *pFichero;
 struct DatosPersonales persona;

 pFichero = fopen(nombreFichero,
"rb"); /* abre fichero para lectura */
```

```
 /* Lee datos del fichero hasta que
llega al final
    Tambien serviría un while
(fscanf(...)!=EOF)*/
 while (fscanf(pFichero, "%ld %s %s
%f", &persona.dni, persona.nombre,
                persona.apellido,
&persona.salario)==NUM_CAMPOS)
 {
    escribirDatosPersonales(persona);
 }
 fclose(pFichero);
}
```

```
 /*Lee datos del fichero hasta que
llega al final */
 while (fgets(linea, MAX_LINEA,
pFichero)!=NULL)
  {
    //Procesamos dni
    sscanf(linea, "%ld",
&persona.dni);

    //Procesamos nombre
    fgets(linea, MAX_LINEA, pFichero);
    limpiarLinea(linea);
    strcpy(persona.nombre, linea);

    //Procesamos apellidos - se
admiten espacios en blanco
    fgets(linea, MAX_LINEA, pFichero);
    limpiarLinea(linea);
    strcpy(persona.apellido, linea);

          //Procesamos salario
    fgets(linea, MAX_LINEA, pFichero);
    sscanf(linea, "%f",
&persona.salario);

    escribirDatosPersonales(persona);
  }
 fclose(pFichero);
}
```

```
 /* Se abre para lectura el fichero
original */
 pFichero1 = fopen(nombreFichero,
"r");

 /* Fichero temporal para volcar los
registros que no se borran */
 pFichero2 = fopen("temporal", "w");

 /* Se recorre el fichero original y
los registros que no hay que borrar
    se pasan al fichero temporal */

  while (fgets(linea, MAX_LINEA,
pFichero1)!=NULL)
  { //Procesamos dni
    sscanf(linea, "%ld", persona.dni);

    //Procesamos nombre
    fgets(linea, MAX_LINEA,pFichero1);
    limpiarLinea(linea);
    strcpy(persona.nombre, linea);

    //Procesamos apellidos - se
admiten espacios en blanco
    fgets(linea, MAX_LINEA,pFichero1);
    limpiarLinea(linea);
    strcpy(persona.apellido, linea);

    //Procesamos salario
    fgets(linea, MAX_LINEA,pFichero1);
    sscanf(linea, "%f",
&persona.salario);

 //El registro no contiene dni_buscar
    if (persona.dni != dni_buscar)
    fprintf(pFichero2,
"%ld\n%s\n%s\n%.3f\n", persona.dni,
persona.nombre, persona.apellido,
persona.salario);
    else
      borrado = 1;
  }

  /* Se cierran los ficheros */
  fclose(pFichero1);
  fclose(pFichero2);

  /* Se borra el fichero original */
  remove(nombreFichero);

  /* Se renombra el temporal con el
nombre del original */
  rename("temporal", nombreFichero);
  return(borrado);
}
```

```
 /* lee datos del fichero uno a uno
hasta que llega al final */
 while(fread(&persona,sizeof(struct
DatosPersonales),1,pFichero) == 1)
  {
   /* Hay que comprobar que no esté
marcado como borrado */
   if (persona.dni != BORRADO)
    escribirDatosPersonales(persona);
/* Muestra el registro por pantalla */
  }
 fclose(pFichero);
}
```

long contarRegistros(char *fichero)

```
long contarRegistros(char
*nombreFichero)
{
 FILE *pFichero;
 long nRegistros=0;
 struct DatosPersonales persona;
 /* abre fichero paa lectura */
 pFichero = fopen(nombreFichero, "r");

 /* Lee datos del fichero hasta que
llega al final
    Tambien serviría un while
(fscanf(...)!=EOF) */
  while (fscanf(pFichero, "%ld %s %s
%f", &persona.dni, persona.nombre,
                persona.apellido,
&persona.salario)==NUM_CAMPOS)
  {
    nRegistros++;
  }

 fclose(pFichero);
 return(nRegistros);
}
```

```
long contarRegistros(char*
nombreFichero)
{
 FILE *pFichero;
 long nRegistros=0;
 struct DatosPersonales persona;
 char linea[MAX_LINEA];
 /* Abre fichero para lectura */
 pFichero = fopen(nombreFichero, "r");
 /* lee datos del fichero hasta que
llega al final */
 while (fgets(linea, MAX_LINEA,
pFichero)!=NULL)
  {
    nRegistros++;
  }
 fclose(pFichero);
 return(nRegistros/NUM_CAMPOS);
}
```

```
long contarRegistros(char*
nombreFichero)
{
 FILE *pFichero;
 long numeroRegistros;

 /* Se abre para lectura el fichero */
 pFichero = fopen(nombreFichero,
"rb");

 /* Nos situamos al final del mismo */
 fseek(pFichero, 0L, SEEK_END);

 /* ftell devuelve el numero de bytes
desde el principio del fichero
    hasta la posicion actual que es el
final del fichero. */
 numeroRegistros =
ftell(pFichero)/sizeof(struct
DatosPersonales);
 fclose(pFichero);
 return numeroRegistros;
}
```

```
long contarRegistros(char
*nombreFichero)
{
 FILE *pFichero;
 struct DatosPersonales  persona;
 long numeroRegistros=0;

 /* Se abre para lectura el fichero */
 pFichero = fopen(nombreFichero,
"rb");

 /* Nos situamos al final del mismo */
 while(fread(&persona,sizeof(struct
DatosPersonales),1,pFichero) == 1)
 {
   /*Usar fseek y ftell contaria
tambien los marcados para borrar*/
   /*solo contabilizamos si el
registro no esta marcado para borrar*/
   if(persona.dni !=BORRADO)
      numeroRegistros++;
 }
 fclose(pFichero);
 return numeroRegistros;
}
```

void anadirRegistro(char *fichero, struct DatosPersonales persona)

```
void anadirRegistro(char
*nombreFichero, struct DatosPersonales
persona)
{
 FILE *pFichero;
 /* abre el fichero para anadir */
 pFichero = fopen(nombreFichero, "a");

 /* guarda los datos en el fichero */
 fprintf(pFichero, "%ld %s %s %.3f\n",
persona.dni, persona.nombre,
        persona.apellido,
persona.salario);
 fclose(pFichero);
}
```

```
void anadirRegistro(char*
nombreFichero, struct DatosPersonales
persona)
{
 FILE *pFichero;
 /* abre el fichero para anadir */
 pFichero = fopen(nombreFichero, "a");

 /* escribe los datos en el fichero */
 fprintf(pFichero,
"%ld\n%s\n%s\n%.3f\n", persona.dni,
persona.nombre,
         persona.apellido,
persona.salario);
 fclose(pFichero);
}
```

```
void anadirRegistro(char* nombreFichero, struct DatosPersonales persona)
{
 FILE *pFichero;

 pFichero = fopen(nombreFichero, "ab"); /* abre el fichero para anadir */
 /* guarda los datos en el fichero */
 fwrite(&persona, sizeof(struct DatosPersonales), 1, pFichero);
 fclose(pFichero);
}
```

struct DatosPersonales registro_i(char *fichero, int i)

```
struct DatosPersonales registro_i(char
*nombreFichero, long i)
{
 FILE *pFichero;
 long j;
 struct DatosPersonales persona;

 pFichero = fopen(nombreFichero, "r");

 for(j=0; j<i; j++)
 {
```

```
struct DatosPersonales registro_i(char
*nombreFichero, long i)
{
 FILE *pFichero;
 long j;
 struct DatosPersonales persona;
 char linea[MAX_LINEA];

 pFichero = fopen(nombreFichero, "r");

 for(j=0; j<i-1; j++)
```

```
struct DatosPersonales
registro_i(char* nombreFichero, long
i)
{
 FILE *pFichero;
 struct DatosPersonales aux;

 /* Se abre para lectura el fichero */
 pFichero = fopen(nombreFichero,
"rb");
```

```
struct DatosPersonales registro_i(char
*nombreFichero, long i)
{
 FILE *pFichero;
 long cont=0;
 struct DatosPersonales persona;

 /* Se abre para lectura el fichero */
 pFichero = fopen(nombreFichero,
"rb");
```

```
    fscanf(pFichero, "%ld %s %s %f",
&persona.dni, persona.nombre,
          persona.apellido,
&persona.salario);
 }
 fclose(pFichero);

 /* devuelve el registro leido */
 return persona;
}
```

```
 {
     fgets(linea, MAX_LINEA,
pFichero);
     fgets(linea, MAX_LINEA,
pFichero);
     fgets(linea, MAX_LINEA,
pFichero);
     fgets(linea, MAX_LINEA,
pFichero);
 }

 //Procesamos dni
 fgets(linea, MAX_LINEA, pFichero);
 sscanf(linea, "%ld", &persona.dni);

 //Procesamos nombre
 fgets(linea, MAX_LINEA, pFichero);
 limpiarLinea(linea);
 strcpy(persona.nombre, linea);

 //Procesamos apellidos - se admiten
espacios en blanco
 fgets(linea, MAX_LINEA, pFichero);
 limpiarLinea(linea);
 strcpy(persona.apellido, linea);

 //Procesamos salario
 fgets(linea, MAX_LINEA, pFichero);
 sscanf(linea, "%f",
&persona.salario);

 fclose(pFichero);

 /* devuelve el registro leido */
 return persona;
}
```

```
 /* Nos desplazamos al final del
registro i-1, desde el principio
 del fichero con la funcion fseek*/
 fseek(pFichero, (i-1)*sizeof(struct
DatosPersonales), SEEK_SET);

 /* Se lee el siguiente registro */
 fread(&aux,sizeof(struct
DatosPersonales),1,pFichero);

 fclose(pFichero);
 /* devuelve el registro leido */
 return aux;
}
```

```
 while(cont<i)
 {
   /*Usar fseek y ftell contaria
tambien los marcados para borrar*/
   fread(&persona,sizeof(struct
DatosPersonales),1,pFichero);
   if(persona.dni !=BORRADO)
      cont++;
 }

 fclose(pFichero);
 /* devuelve el registro leido */
 return persona;
}
```

int buscarporDni(char *fichero, long dni_buscar, struct DatosPersonales *persona)

```
int buscarporDni(char *nombreFichero,
long dni, struct DatosPersonales
*persona)
{
 FILE *pFichero;
 int encontrado = 0; /* variables
auxiliares para la busqueda */
 int cont;
 struct DatosPersonales auxiliar;
 /* abre fichero para lectura */
 pFichero = fopen(nombreFichero, "r");

 /*evaluacion en cortocircuito*/
 while (!encontrado &&
(fscanf(pFichero, "%ld %s %s %f",
        &auxiliar.dni,
auxiliar.nombre, auxiliar.apellido,
              &auxiliar.salario) ==
NUM_CAMPOS))
```

```
int buscarporDni(char *nombreFichero,
long dni_buscar, struct
DatosPersonales *persona)
{
 FILE *pFichero;
 int encontrado = 0;
 int cont;
 char linea[MAX_LINEA];
 struct DatosPersonales aux;

 /* abre fichero para lectura */
 pFichero = fopen(nombreFichero, "r");

 //evaluación en corto circuito
 while (!encontrado && (fgets(linea,
MAX_LINEA, pFichero)!=NULL))
  {
    //Procesamos dni
    sscanf(linea, "%ld", &aux.dni);
```

```
int buscarporDni(char* nombreFichero, long dni, struct DatosPersonales
*persona)
{ FILE *pFichero;
 int encontrado = 0; /* variables auxiliares para la busqueda */
 struct DatosPersonales auxiliar;
 /* abre fichero para lectura */
 pFichero = fopen(nombreFichero, "rb");
 while (fread(&auxiliar, sizeof(struct DatosPersonales), 1, pFichero)==1)
  {  if (auxiliar.dni == dni)
    /* ha encontrado el registro */
     {
      encontrado = 1;
          /* almacena en persona el registro encontrado */
      *persona = auxiliar;
     }
  }
 fclose(pFichero); /* se cierra el fichero */
 return encontrado;
}
```

```
  {
    if (auxiliar.dni == dni) /* ha
encontrado el registro */
     {
      encontrado = 1;
      *persona = auxiliar; /* almacena
en persona el registro encontrado */
     }
   }
 fclose(pFichero); /* se cierra el
fichero */
 return encontrado;
}
```

```
    //Procesamos nombre
    fgets(linea, MAX_LINEA, pFichero);
    limpiarLinea(linea);
    strcpy(aux.nombre, linea);

    //Procesamos apellidos - se
admiten espacios en blanco
    fgets(linea, MAX_LINEA, pFichero);
    limpiarLinea(linea);
    strcpy(aux.apellido, linea);

    //Procesamos el salario
    fgets(linea, MAX_LINEA, pFichero);
    sscanf(linea, "%f", &aux.salario);

          //almacena en persona el
registro encontrado
    if (aux.dni == dni_buscar) /* ha
encontrado el registro */
    {
      encontrado = 1;
      *persona = aux;
    }
  }

 fclose(pFichero); /* se cierra el
fichero */
 return encontrado;
}
```

int mostrarporNombre(char *fichero, char *auxNombre)

```
int mostrarporNombre(char
*nombreFichero, char *auxNombre)
{
 FILE *pFichero;
 struct DatosPersonales auxiliar;
 int encontrado = 0;
 /* abre fichero para lectura */
 pFichero = fopen(nombreFichero, "r");

 /*Evaluacion en cortocircuito
 / Tambien serviría un while
(fscanf(...)!=EOF)*/
 while (!encontrado &&
(fscanf(pFichero, "%ld %s %s %f",
        &auxiliar.dni,
auxiliar.nombre, auxiliar.apellido,
              &auxiliar.salario) ==
NUM_CAMPOS))

  {
   if (strcmp(auxiliar.nombre,
auxNombre) == 0)
   /* se ha encontrado un registro con
ese nombre */
    {
```

```
int mostrarporNombre(char
*nombreFichero, char *auxNombre)
{
 FILE *pFichero;
 struct DatosPersonales persona;
 int encontrado = 0;
 char linea[MAX_LINEA];
 /* abre fichero para lectura */
 pFichero = fopen(nombreFichero, "r");

 while (fgets(linea, MAX_LINEA,
pFichero)!=NULL)
  {
    //Procesamos dni
    sscanf(linea, "%ld",
&persona.dni);

    //Procesamos nombre
    fgets(linea, MAX_LINEA, pFichero);
    limpiarLinea(linea);
    strcpy(persona.nombre, linea);

    //Procesamos apellidos - se
admiten espacios en blanco
    fgets(linea, MAX_LINEA, pFichero);
```

```
int mostrarporNombre(char*
nombreFichero, char *auxNombre)
{
 FILE *pFichero;
 struct DatosPersonales auxiliar;
 int encontrado = 0;
  /* abre fichero para lectura */
pFichero = fopen(nombreFichero, "rb");
 while(fread(&auxiliar,sizeof(struct
DatosPersonales),1,pFichero) == 1)
  {
   if (strcmp(auxiliar.nombre,
auxNombre)==0)
   /* se ha encontrado un registro con
ese nombre */
    {
    escribirDatosPersonales(auxiliar);
/* se escriben sus datos */
     encontrado = 1;
    }
  }
 fclose(pFichero); /* se cierra el
fichero */
 return encontrado;
}
```

```
int mostrarporNombre(char
*nombreFichero, char *auxNombre)
{ FILE *pFichero;
 struct DatosPersonales auxiliar;
 int encontrado = 0;
 /* abre fichero para lectura */
 pFichero = fopen(nombreFichero, rb");
 while(fread(&auxiliar,sizeof(struct
DatosPersonales),1,pFichero) == 1)
 /* se leen todos los registros */
  {
   if (strcmp(auxiliar.nombre,
auxNombre) == 0
       && auxiliar.dni!=BORRADO)
   /* se ha encontrado un registro con
ese nombre y no está marcado */
    {
escribirDatosPersonales(auxiliar);
/* se escriben sus datos */
     encontrado = 1;
    }
  }
fclose(pFichero); /* se cierra el
fichero */
 return encontrado;}
```

```
escribirDatosPersonales(auxiliar); /*
se escriben sus datos */
     encontrado = 1;
    }
  }
 fclose(pFichero); /* se cierra el
fichero */
 return encontrado;
}
```

```
    limpiarLinea(linea);
    strcpy(persona.apellido, linea);

    //Procesamos el salario
    fgets(linea, MAX_LINEA, pFichero);
    sscanf(linea, "%f",
&persona.salario);

    //se ha encontrado un registro con
ese nombre
    if (strcmp(persona.nombre,
auxNombre) == 0)
    {

escribirDatosPersonales(persona); /*
se escriben sus datos */
                        encontrado = 1;
    }
  }

 fclose(pFichero); /* se cierra el
fichero */
 return encontrado;
}
```

int actualizarporDni(char* fichero, long dni)

```
int actualizarporDni(char*
nombreFichero, long dni)
{
  FILE *pFichero1, *pFichero2;
  struct DatosPersonales aux;
  int encontrado = 0;

 /* Se abre para lectura el fichero
original */
 pFichero1 = fopen(nombreFichero,
"r");

 /* Fichero temporal para volcar los
registros que no se borran */
 pFichero2 = fopen("temporal", "w");

 /* Recorre el fichero original y los
registros que no hay que borrar
    se pasan al fichero temporal.
    Tambien serviría un while
(fscanf(...)!=EOF)*/
 while(fscanf(pFichero1, "%ld %s %s
%f", &aux.dni, aux.nombre,
       aux.apellido, &aux.salario) ==
NUM_CAMPOS)
 {
   if (aux.dni == dni)
   {
         aux =
introducirDatosPersonales();
         encontrado = 1;
```

```
int actualizarporDni(char*
nombreFichero, long dni_buscar)
{
 FILE *pFichero1, *pFichero2;
 struct DatosPersonales persona;
 char linea[MAX_LINEA];
 int encontrado = 0;

 /* Se abre para lectura el fichero
original */
 pFichero1 = fopen(nombreFichero,
"r");

 /* Fichero temporal para volcar los
registros que no se borran */
 pFichero2 = fopen("temporal", "w");

 /* Se recorre el fichero original y
los registros que no hay que borrar
    se pasan al fichero temporal */

 while ((fgets(linea, MAX_LINEA,
pFichero1)!=NULL))
  {
    //Procesamos el dni
    sscanf(linea, "%ld",
&persona.dni);

    //Procesamos el nombre
    fgets(linea, MAX_LINEA,
pFichero1);
```

```
int actualizarporDni(char* nombreFichero, long dni)
{
 FILE *pFichero;
 struct DatosPersonales aux;
 int encontrado =0;
  /* Se abre para lectura y escritura */
 pFichero = fopen(nombreFichero, "r+b");

 /* Se recorre el fichero */
 while(fread(&aux, sizeof(struct DatosPersonales),1,pFichero) == 1 )
  {
   /* Se comprueba si el registro tiene ese dni*/
   if (aux.dni == dni)
   {       aux = introducirDatosPersonales();
           encontrado = 1;

    /* Nos situamos al principio del registro encontrado */
     fseek(pFichero, -(int)sizeof(struct DatosPersonales), SEEK_CUR);
     /* Se reescribe el registro */
     fwrite(&aux, sizeof(struct DatosPersonales), 1, pFichero);
     //La siguiente orden es necesario para que en windows
     //se actualicen los flags de FILE*
     //En Linux, no es necesario, pero se puede dejar
     //El estándar requiere fflush para hacer fread despues de fwrite
     fflush(pFichero);
    }
  }
  /* Se cierra el fichero*/
  fclose(pFichero);
  return(encontrado)
}
```

```
   }
   fprintf(pFichero2, "%ld %s %s
%.3f\n", aux.dni, aux.nombre,
           aux.apellido, aux.salario);
 }

  /* Se cierran los ficheros */
  fclose(pFichero1);
  fclose(pFichero2);

 /* Se borra el fichero original */
 remove(nombreFichero);

 /* Se renombra el temporal con el
nombre del original */
 rename("temporal", nombreFichero);

 return(encontrado);
}
```

```
    limpiarLinea(linea);
    strcpy(persona.nombre, linea);

    //Leemos apellidos - se admiten
espacios en blanco
    fgets(linea, MAX_LINEA,
pFichero1);
    limpiarLinea(linea);;
    strcpy(persona.apellido, linea);

          //Leemos el salario
    fgets(linea, MAX_LINEA,
pFichero1);
    sscanf(linea, "%f",
&persona.salario);

    if (persona.dni == dni_buscar)
    {
           persona =
introducirDatosPersonales();
           encontrado = 1;
    }
    fprintf(pFichero2,
"%ld\n%s\n%s\n%.3f\n", persona.dni,
persona.nombre,
           persona.apellido,
persona.salario);
  }
  /* Se cierran los ficheros */
  fclose(pFichero1);
  fclose(pFichero2);

 /* Se borra el fichero original */
 remove(nombreFichero);

 /* Se renombra el temporal con el
nombre del original */
 rename("temporal", nombreFichero);
 return(encontrado);
}
```

int borrarporDni(char *fichero, long dni)

```
int borrarporDni(char *nombreFichero,
long dni)
{
 FILE *pFichero1, *pFichero2;
 struct DatosPersonales aux;
 int borrado = 0;

 /* Se abre para lectura el fichero
original */
 pFichero1 = fopen(nombreFichero,
"r");

 /* Fichero temporal para volcar los
registros que no se borran */
 pFichero2 = fopen("temporal", "w");
```

```
int borrarporDni(char *nombreFichero,
long dni_buscar)
{
 FILE *pFichero1, *pFichero2;
 struct DatosPersonales persona;
 char linea[MAX_LINEA];
 int borrado = 0;

 /* Se abre para lectura el fichero
original */
 pFichero1 = fopen(nombreFichero,
"r");

 /* Fichero temporal para volcar los
registros que no se borran */
```

```
int borrarporDni(char* nombreFichero,
long dni_buscar)
{
 FILE *pFichero1, *pFichero2;
 struct DatosPersonales aux;
 int borrado=0;

 /* Se abre para lectura el fichero
original */
 pFichero1 = fopen(nombreFichero,
"rb");

 /* Fichero temporal para volcar los
registros que no se borran */
 pFichero2 = fopen("temporal", "wb");
```

```
int borrarporDni(char *nombreFichero,
long dni)
{
 FILE *pFichero;
 struct DatosPersonales aux;
 int encontrado = 0;

 /* Se abre para lectura y escritura
el fichero */
 pFichero = fopen(nombreFichero,
"r+b");

 /* Se recorre el fichero original y
los registros a borrar se
    reescriben marcandolos como
```

```
 /* Recorre el fichero original y los
registros que no hay que borrar
    se pasan al fichero temporal
    Tambien serviría un while
(fscanf(...)!=EOF)*/
 while(fscanf(pFichero1, "%ld %s %s
%f",  &aux.dni, aux.nombre,
       aux.apellido, &aux.salario) ==
NUM_CAMPOS)
  {
   if (aux.dni != dni)
    fprintf(pFichero2, "%ld %s %s
%.3f\n", aux.dni, aux.nombre,
aux.apellido, aux.salario);
   else
    borrado = 1;
  }

  /* Se cierran los ficheros */
  fclose(pFichero1);
  fclose(pFichero2);

 /* Se borra el fichero original */
 remove(nombreFichero);

 /* Se renombra el temporal con el
nombre del original */
 rename("temporal", nombreFichero);
 return(borrado);
}
```

```
 pFichero2 = fopen("temporal", "w");

 /* Se recorre el fichero original y
los registros que no hay que borrar
    se pasan al fichero temporal */

  while (fgets(linea, MAX_LINEA,
pFichero1)!=NULL)
  {
    //Procesamos dni
    sscanf(linea, "%ld",
&persona.dni);

    //Procesamos nombre
    fgets(linea, MAX_LINEA,
pFichero1);
    limpiarLinea(linea);
    strcpy(persona.nombre, linea);

    //Procesamos apellidos - se
admiten espacios en blanco
    fgets(linea, MAX_LINEA,
pFichero1);
    limpiarLinea(linea);
    strcpy(persona.apellido, linea);

    //Procesamos salario
    fgets(linea, MAX_LINEA,
pFichero1);
    sscanf(linea, "%f",
&persona.salario);

          //El registro no contiene
dni_buscar
    if (persona.dni != dni_buscar)
    fprintf(pFichero2,
"%ld\n%s\n%s\n%.3f\n", persona.dni,
persona.nombre,
           persona.apellido,
persona.salario);
    else
      borrado = 1;
  }

  /* Se cierran los ficheros */
  fclose(pFichero1);
  fclose(pFichero2);

  /* Se borra el fichero original */
  remove(nombreFichero);

  /* Se renombra el temporal con el
nombre del original */
  rename("temporal", nombreFichero);
  return(borrado);
}
```

```
 /* Se recorre el fichero original y
los registros que no hay que borrar
    se pasan al fichero temporal */

 while(fread(&aux, sizeof(struct
DatosPersonales),1,pFichero1) == 1 )
  {
   /* Se comprueba si el registro
tiene ese dni*/
   if (dni_buscar!=aux.dni)
     fwrite(&aux, sizeof(struct
DatosPersonales), 1, pFichero2);
   else
     borrado = 1;
  }
  /* Se cierran los ficheros */
  fclose(pFichero1);
  fclose(pFichero2);

 /* Se borra el fichero original */
 remove(nombreFichero);

 /* Se renombra el temporal con el
nombre del original */
 rename("temporal", nombreFichero);
 return(borrado);
}
```

```
borrados */
 while((fread(&aux, sizeof(struct
DatosPersonales),1,pFichero) == 1
)&&!encontrado)
  {
   /* Se comprueba si el registro
tiene ese nombre y no está marcado */
   if (aux.dni == dni)
    {
     /* Se marca como borrado */
     aux.dni = BORRADO;
     /* Nos situamos al principio del
registro encontrado */
     fseek(pFichero, -
(int)sizeof(struct DatosPersonales),
SEEK_CUR);
     /* Se reescribe el registro */
     fwrite(&aux, sizeof(struct
DatosPersonales), 1, pFichero);
     //La siguiente orden es necesario
para que en windows
     //se actualicen los flags de
FILE*
     //En Linux, no es necesario, pero
se puede dejar
     //El estándar requiere fflush
para hacer fread despues de fwrite
     fflush(pFichero);
     /* Se ha borrado al menos un
registro */
     encontrado = 1;
    }
  }
  /* Se cierra el fichero */
  fclose(pFichero);
  retur
```

struct DatosPersonales* ficheroAVector(char* fichero, long* nEle)

```
struct DatosPersonales*
ficheroAVector(char* nombreFichero,
long* nEle)
{
   FILE *pFichero;
   struct DatosPersonales persona, *V;
   long i=0;

   * nEle =
contarRegistros(nombreFichero);
   V = reservarVector(*nEle);
   pFichero = fopen(nombreFichero,
"r");

   /*Tambien serviría un while
(fscanf(...)!=EOF)*/
   while (fscanf(pFichero, "%ld %s %s
%f", &persona.dni, persona.nombre,
persona.apellido,

&persona.salario)==NUM_CAMPOS)
   {
      V[i]=persona;
      i++;
   }

   fclose(pFichero);
   return(V);
}
```

```
struct DatosPersonales*
ficheroAVector(char* nombreFichero,
long* nEle)
{
   FILE *pFichero;
   struct DatosPersonales persona, *V;
   long i=0;
   char linea[MAX_LINEA];

   *nEle =
contarRegistros(nombreFichero);
   V = reservarVector(*nEle);
   pFichero = fopen(nombreFichero,
"r");

   /*Leemos todo el fichero*/
   while (fgets(linea, MAX_LINEA,
pFichero)!=NULL)
   {
    //Procesamos dni
    sscanf(linea, "%ld",
&persona.dni);

    //Procesamos nombre
    fgets(linea, MAX_LINEA, pFichero);
    limpiarLinea(linea);
    strcpy(persona.nombre, linea);

    //Procesamos apellidos - se
admiten espacios en blanco
    fgets(linea, MAX_LINEA, pFichero);
    limpiarLinea(linea);
    strcpy(persona.apellido, linea);

    //Procesamos salario
    fgets(linea, MAX_LINEA, pFichero);
    sscanf(linea, "%f",
&persona.salario);

    V[i]=persona;
    i++;
   }

   fclose(pFichero);
   return(V);
}
```

```
struct DatosPersonales*
ficheroAVector(char* nombreFichero,
long* nEle)
{
   struct DatosPersonales* V;
   FILE* pFichero;

   *nEle =
contarRegistros(nombreFichero);
   V = reservarVector(*nEle);
   pFichero = fopen(nombreFichero,
"rb");

   /*Leemos todo el fichero (mas
rapido que elemento a elemento)*/
   fread(V, sizeof(struct
DatosPersonales),*nEle,pFichero);
   fclose(pFichero);
   return(V);
}
```

```
struct DatosPersonales*
ficheroAVector(char* nombreFichero,
long* nEle)
{
   struct DatosPersonales* V;
   FILE* pFichero;
   long registros, i;

   //Contamos registros, incluidos los
borrados
   //contarRegistros solo cuenta los
activos, por lo que no es valida
   pFichero = fopen(nombreFichero,
"rb");
   fseek(pFichero, 0, SEEK_END);
   registros =
ftell(pFichero)/sizeof(struct
DatosPersonales);

   V = reservarVector(registros);

   /*Leemos todo el fichero (mas
rapido que elemento a elemento)*/
   fseek(pFichero, 0, SEEK_SET);
   fread(V, sizeof(struct
DatosPersonales), registros,
pFichero);
   fclose(pFichero);

   //Nos quedamos solamente con los
registros no borrados
   for(i=0, *nEle=0; i<registros; i++)
   {
      printf("\ncopiando registro: %ld
nEle: %ld", i, *nEle);
          if(V[i].dni!=BORRADO)
          {
                V[*nEle] = V[i];
            (*nEle)++; //Parentesis
necesarios
      }
   }
   V= (struct DatosPersonales*)
realloc(V, (*nEle)*sizeof(struct
DatosPersonales));
   return(V);
}
```

void vectorAFichero(char* fichero, struct DatosPersonales* V, long nEle)

```
void vectorAFichero(char*
nombreFichero, struct DatosPersonales*
V, long nEle)
{
   FILE* pFichero;
   long i=0;
```

```
void vectorAFichero(char*
nombreFichero, struct DatosPersonales*
V, long nEle)
{
   FILE* pFichero;
   long i=0;
```

```
void vectorAFichero(char* nombreFichero, struct DatosPersonales* V, long nEle)
{   FILE* pFichero;
    pFichero = fopen(nombreFichero, "wb");
   fwrite(V, sizeof(struct DatosPersonales), nEle, pFichero);
   fclose(pFichero);
}
```

```
   pFichero = fopen(nombreFichero,
"w");
   for (i=0; i<nEle; i++)
   fprintf(pFichero, "%ld %s %s
%.3f\n", V[i].dni, V[i].nombre,
V[i].apellido,
             V[i].salario);
   fclose(pFichero);
}
```

```
   pFichero = fopen(nombreFichero,
"wt");
   for (i=0; i<nEle; i++)
   fprintf(pFichero,
"%ld\n%s\n%s\n%.3f\n", V[i].dni,
V[i].nombre,
         V[i].apellido, V[i].salario);
   fclose(pFichero);
}
```

int borrarMarcados(char *fichero)

```
int borrarMarcados(char
*nombreFichero);


int main()
{

  char nombreFichero[15]; /* variable
para el nombre del fichero */
  int borrados ;
   /* Introduccion del nombre del
fichero para trabajar */
  printf(" Nombre del fichero: ");
  scanf("%s", nombreFichero);

  if(existeFichero(nombreFichero))
  {
     borrados =
borrarMarcados(nombreFichero);
     printf("\nBorrado Fisico de <%d>
registros", borrados);
  }
  else
  {
     printf("\nEl archivo no existe");
  }

  return 0;
}
```

struct DatosPersonales introducirDatosPersonales()

```
struct DatosPersonales introducirDatosPersonales()
{
 struct DatosPersonales aux;

 /*Introduccion de datos */
 printf("DNI : ");
 scanf("%ld", &aux.dni);
 getchar();

 printf("Nombre : ");
 fgets(aux.nombre, MAX_LINEA, stdin);
 limpiarLinea(aux.nombre);

 printf("Apellido :");
 fgets(aux.apellido, MAX_LINEA, stdin);
 limpiarLinea(aux.apellido);

 printf("Salario:");
 scanf("%f", &aux.salario);
 getchar();

 /* devolucion de los datos de la persona */
 return aux;
}
```

void escribirDatosPersonales(struct DatosPersonales aux)

```
void escribirDatosPersonales(struct DatosPersonales aux)
{
 printf("DNI: %ld\n\tNombre: <%s>  Apellido(s): <%s>  Salario: %.3f\n",
        aux.dni, aux.nombre, aux.apellido, aux.salario);
}
```

int existeFichero(char *nombreFichero)

```
int existeFichero(char *nombreFichero)
{
 FILE *pFichero;

 pFichero = fopen(nombreFichero, "r"); /* abre fichero para lectura */
 if (pFichero == NULL) /* el fichero no existe */
  {
   return 0;
  }
 else /* el fichero existe */
  {
   fclose (pFichero);
   return 1;
  }
}
```

long contarBytes(char *nombreFichero)

```
long contarBytes(char *nombreFichero)
{ FILE* f;
  long tam;

  if((f=fopen(nombreFichero, "r"))==NULL)
  {
    fprintf(stderr, "\nError: no puedo abrir <%s>", nombreFichero);exit(-1);
  }
  if(fseek(f, 0L, SEEK_END))
  {
    fprintf(stderr, "\nError: no puedo usar <%s>", nombreFichero); exit(-1);
  }
  tam = ftell(f);
  fclose(f);
  return(tam);
}
```

DatosPersonales* reservarVector(long nEle)

```
struct DatosPersonales* reservarVector(long nEle)
{       struct DatosPersonales* V=NULL;

        if((V=(struct DatosPersonales*)malloc(nEle*sizeof(struct DatosPersonales)))==NULL)
        {       printf("\nError: No se ha podido reservar la memoria");
                exit(-1);
        }
        return(V);
}
```

void liberarVector(struct DatosPersonales V)**

```
void liberarVector(struct DatosPersonales** V)
{
   free(*V);
   *V=NULL;
}
```

void incrementarSalarios(char* nombreFichero, float incremento)

```
void incrementarSalarios(char* nombreFichero, float incremento)
{
   struct DatosPersonales * V;
   long nEle;
   long i;

   /*Para reducir los accesos a disco trabajamos con un vector*/
   V = ficheroAVector(nombreFichero, &nEle);

   /*Actualizamos los datos en el vector*/
   for(i=0; i<nEle; i++)
     V[i].salario += incremento;

   /*Escribimos el vector en el fichero*/
   vectorAFichero(nombreFichero, V, nEle);

   /*liberamos el vector*/
   liberarVector(&V);
}
```

void limpiarLinea(char* linea)

```
void limpiarLinea(char* linea)
{
  if(linea[strlen(linea)-1]=='\n')
     linea[strlen(linea)-1] = '\0';
}
```

void escribirDatosPersonales(struct DatosPersonales aux)

```
void escribirDatosPersonales(struct DatosPersonales aux)
{
 printf("DNI: %ld\n\tNombre: %s  Apellido: %s  Salario: %.3f\n",
         aux.dni, aux.nombre, aux.apellido, aux.salario);
}
```

int existeFichero(char *fichero)

```
int existeFichero(char *fichero)
{
 FILE *pFichero;

 pFichero = fopen(fichero, "r"); /* abre fichero para lectura */
 if (pFichero == NULL) /* el fichero no existe */
  {
   return 0;
  }
 else /* el fichero existe */
  {
   fclose (pFichero);
   return 1;
  }
}
```

long contarBytes(char *fichero)

```
long contarBytes(char *fichero)
{
  FILE* f;
  long tam;

  if((f=fopen(fichero, "r"))==NULL)
  {
    fprintf(stderr, "\nError: no puedo abrir <%s>", fichero);exit(-1);
  }
  if(fseek(f, 0L, SEEK_END))
  {
    fprintf(stderr, "\nError: no puedo usar <%s>", fichero); exit(-1);
  }
  tam = ftell(f);
  fclose(f);
  return(tam);
}
```

struct DatosPersonales* reservarVector(long nEle)

```
struct DatosPersonales* reservarVector(long nEle)
{
   struct DatosPersonales* V=NULL;

   if((V=(struct DatosPersonales*)malloc(nEle*sizeof(struct DatosPersonales)))==NULL)
   {
      printf("\nError: No se ha podido reservar la memoria");
      exit(-1);

   return(V);
}
```

void liberarVector(struct DatosPersonales V)**

```
void liberarVector(struct DatosPersonales** V)
{
   free(*V);
   *V=NULL;
}
```

void incrementarSalarios(char* fichero, float incremento)

```
void incrementarSalarios(char* fichero, float incremento)
{
   struct DatosPersonales * V;
   long nEle;
   FILE* pFichero;
   long i;

   /*Para reducir los accesos a disco trabajamos con un vector*/
   V = ficheroAVector(fichero, &nEle);

   /*Actualizamos los datos en el vector*/
   for(i=0; i<nEle; i++)
     V[i].salario += incremento;

   /*Escribimos el vector en el fichero*/
   vectorAFichero(fichero, V, nEle);

   /*liberamos el vector*/
   liberarVector(&V);
}
```