Ejercicio 3

En un fichero en disco disponemos del nombre y del dni de un conjunto de alumnos. La estructura de cada registro del fichero es así:

```
tlpedef struct
{
char nombre[60];
unsigned. long dni;
} alumno;
```

Se desea escribir un programa para visualizar los registros del fichero ordenados por el miembro dni. Para ello leeremos los registros del fichero y los almacenaremos en un árbol binario de búsqueda ordenado por el dni. Cada nodo del árbol será de la forma siguiente:

```
typedef struct elem
{
afumno datos; /* datos del nodo */
struct elem *izdo; /* puntero al subárbo1 izquierdo */
struct elem *dcho; /* puntero al subárbo1 derecho */
} nodo;
```

Se pide:

a) Escribir una función *insertar* que permita añadir nodos a una estructura en árbol binario de búsqueda. Los nodos estarán ordenados por el miembro dni.

```
nodo *insertar(nodo *raiz, alumno a);
```

El parámetro raíz es la raíz del árbol y a es el registro leído del fichero que hay que añadir al árbol.

b) Escribir una función *visu_ascen* para que recorra el árbol referenciado por raíz y visualice los datos en orden ascendente del miembro dni.

```
void visu_ascen(nodo *raiz) ;
```

b) Escribir una función *visu_descen* para que recorra el árbol referenciado por raíz y visualice los datos en orden descendente del miembro dni.

```
void visu_descen (nodo *raiz ) ;
```

Utilizando las funciones anteriores, escribir un programa *listar* que reciba a través de la línea de órdenes el nombre de un fichero y el orden de presentación, y visualice los registros del fichero en el orden especificado:

```
listar -a fichero
1ístar -d fichero
```

donde *fichero* es el nombre del fichero cuyos regisffos queremos visualizar, ‘a’ significa ascendentemente y ‘d’ significa descendentemente.

```c
/* Arbol binario de búsqueda. Cada nodo hace referencia a una
 * estructura alumno. El árbol está ordenado por el miembro dni.
 * ejercicio3.c
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct
{
  char nombre[60];
  unsigned long dni;
} alumno;

typedef struct elem
{
  alumno datos;       // datos del nodo
  struct elem *izdo;  // puntero al subárbol izquierdo
  struct elem *dcho;  // puntero al subárbol derecho
} nodo;

nodo *insertar(nodo *raiz, alumno a);
void visu_ascen(nodo *raiz);
void visu_descen(nodo *raiz);
nodo *crear_nodo(void);
void borrar_arbol(nodo *a);

int main(int argc, char *argv[])
{
  FILE *pf = NULL;
  nodo *raiz = NULL;
  alumno a;

  if (argc != 3 || (argv[1][1] != 'a' && argv[1][1] != 'd'))
  {
    // Comprobación de la sintaxis de llamada
    printf("Sintaxis:\n\tPara mostrar ascendentemente:\n%s -a
nombre_fichero\n\t"
      "Para mostrar descendentemente:\n%s -d nombre_fichero\n",
argv[0], argv[0]);
    exit(-1);
  }

  // Abrir el fichero
  if ((pf = fopen(argv[2], "rb")) == NULL)
  {
    perror(argv[2]);
    exit(-1);
  }

  fread(&a, sizeof(alumno), 1, pf);
  while ( !feof(pf) && !ferror(pf) )
  {
    raiz = insertar( raiz, a ); // insertar uno a uno, todos los datos
    fread(&a, sizeof(alumno), 1, pf);
  }

  if (argv[1][1] == 'a')
    visu_ascen(raiz);
  else
```

```
    visu_descen(raiz);

  borrar_arbol(raiz);

  return 0;
}

nodo *insertar(nodo *raiz, alumno a)
{
  // Inserta un nuevo alumno

  if (raiz == NULL)  // el nodo con clave x, no está en el árbol
  {               // Insertarlo
    raiz = crear_nodo();
    raiz->datos  = a;
    raiz->izdo = raiz->dcho = NULL;
  }
  else
  {
    if (a.dni < raiz->datos.dni)
              // el valor buscado está a la izquierda de este nodo
      raiz->izdo = insertar(raiz->izdo, a);
    else
    {
      if (a.dni > raiz->datos.dni)
            // el valor buscado está a la derecha de este nodo
        raiz->dcho = insertar(raiz->dcho, a);
    }
  }

  return raiz;
}

void visu_ascen(nodo *raiz)
{
  // Visualiza el árbol ascendentemente
  if (raiz != NULL)
  {
    visu_ascen(raiz->izdo);
    printf("%s (%d)\n", raiz->datos.nombre, raiz->datos.dni);
    visu_ascen(raiz->dcho);
  }
}

void visu_descen(nodo *raiz)
{
  // Visualiza el árbol descendentemente
  if (raiz != NULL)
  {
    visu_descen(raiz->dcho);
    printf("%s (%d)\n", raiz->datos.nombre, raiz->datos.dni);
    visu_descen(raiz->izdo);
  }
}

nodo *crear_nodo()
{
  // Crear un nodo del árbol binario
  nodo *aux;

  if ((aux = (nodo *)malloc(sizeof(nodo))) == NULL)
```

```
  {
    perror("crear_nodo");
    exit(-1);
  }
  aux->izdo = aux->dcho = NULL;
  return aux;
}

void borrar_arbol(nodo *a)
{
  // Liberar la memoria asignada a cada uno de los nodos del árbol
  // direccionado por a. Se recorre el árbol en postorden.

  if (a != NULL)
  {
    borrar_arbol(a->izdo);
    borrar_arbol(a->dcho);
    free(a);
  }
}
```