Ejercicio 4

El filtro sort de Windows lee líneas de texto del fichero estándar de entrada y las muestra en orden alfabético en el fichero estándar de salida. El ejemplo siguiente aclara como funciona el filtro sort:

```
sort[Entrar]
1o que puede hacerse
en cualquier momento
no se hará
en ninqún momento.

(eof)
en cualquier momento
en ninqún momento.
lo que puede hacerse
no se hará
```

Se desea escribir un programa que actúe como el filtro sort. Para ordenar las distintas líneas vamos a ir insertándolas en un árbol binario de búsqueda, de tal forma que al recorrerlo podamos presentar las líneas en orden alfabético. Cada nodo del árbol se ajusta a la definición siguiente:

```
tslpedef struct datos
{
char *linea; /* puntero a una fínea de texto */
struct datos *izq, *der;
] nodo;
```

Para realizar esta aplicación se pide escribir al menos las funciones siguientes:

a) Una función que lea líneas del fichero estándar de entrada y genere un árbol binario de búsqueda. El prototipo de esta función será así:

```
nodo *crear_arbol (void) ;
```

La función devolverá un puntero al nodo raíz del árbol creado.

b) Una función que recorra un árbol de las características anteriores y presente las líneas de texto que referencian sus nodos. El prototipo de esta función será:

```
void imprimir_arbol (nodo *a, char orden) ;
```

Los valores posibles del parámetro orden son: d, presentar las líneas en orden alfabético ascendente, y b, presentar las líneas en orden alfabético descendente.

Escribir un programa de nombre ordenar que responda a la funcionalidad siguiente:

- Lee líneas del fichero estándar de entrada y las presenta en el fichero estándar de salida en orden alfabético ascendente.

`nombre_programa`

- Lee líneas del fichero estándar de entrada y las presenta en el fichero estándar de salida en orden alfabético descendente.

`nombre_programa –r`

```c
/* Ordenación alfabética de frases de un documento
 * procedente de stdin
 * ejercicio4.c
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct datos
{
  char *linea;
  struct datos *izq, *der;
} nodo;

nodo *crear_nodo(void);
nodo *crear_arbol(void);
nodo *poner_nodo(nodo *raiz, char *cadena);
void imprimir_arbol(nodo *a, char orden);
void borrar_arbol(nodo *a);

int main(int argc, char *argv[])
{
  nodo *raiz;

  if( argc > 2 || (argc == 2 && argv[1][1] != 'r'))
  {
    // Comprobación de la sintaxis de llamada
    printf("Sintaxis:\n\tPara mostrar ascendentemente:\n%s\n\t"
      "Para mostrar descendentemente:\n%s -r\n", argv[0], argv[0]);
    exit(-1);
  }
  puts("Escriba varias cadenas de texto. Finalice con EOF.");
  raiz = crear_arbol(); // creación del árbol

  if(argc == 1)
  {
    puts("");
    puts("Las cadenas ordenadas ascendentemente:\n");
    imprimir_arbol(raiz, 'a');
  }
```

```
  else
  {
    puts("");
    puts("Las cadenas ordenadas descendentemente:\n");
    imprimir_arbol(raiz, 'b');
  }

  borrar_arbol(raiz);

  return 0;
}

nodo *crear_arbol(void)
{
  // Crea un árbol según se van tecleando los datos
  nodo *raiz = NULL;
  char cadena[250];

  if (gets(cadena)!= NULL)
    raiz = poner_nodo(raiz, cadena);
  while (gets(cadena) != NULL)
    poner_nodo(raiz, cadena);
  return raiz;
}

nodo *poner_nodo(nodo *raiz, char *cadena)
{
  // Añade un nuevo nodo
  if ( raiz != NULL )
  {
    if ( strcmp( cadena, raiz->linea) < 0 ) // cadena es menor
      raiz->izq = poner_nodo(raiz->izq, cadena);
    else
      raiz->der = poner_nodo( raiz->der, cadena );
  }
  else
  {
    raiz = crear_nodo();
    if ((raiz->linea = (char *) malloc(strlen(cadena) + 1)) == NULL)
    {
      perror("poner_nodo");
      exit(-1);
    }
    strcpy( raiz->linea, cadena );

  }
  return raiz;
}

void imprimir_arbol(nodo *a, char orden)
{
  //clearerr(stdin);
  if(a != NULL)
  {
    if(orden == 'a')
    {
      imprimir_arbol(a->izq, 'a');
      puts(a->linea);
      imprimir_arbol(a->der, 'a');
    }
    else
```

```
    {
      imprimir_arbol(a->der, 'b');
      puts(a->linea);
      imprimir_arbol(a->izq,'b');
    }
  }
}

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

  if ((aux = (nodo *)malloc(sizeof(nodo))) == NULL)
  {
    perror("crear_nodo");
    exit(-1);
  }
  aux->izq = aux->der = 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->izq);
    borrar_arbol(a->der);
    free(a->linea);
    free(a);
  }
}
```