# EJERCICIOS RESUELTOS SOBRE TEMAS DE *PROGRAMACIÓN AVANZADA* EN C

## Versión 1.0 (205008)

Autor: Víctor Sánchez[2]
Web: www.victorsanchez2.net
Correo: webmaster@victorsanchez2.net
victorss19@eresmas.com

Los que posean estos ejercicios pueden hacerme preguntas sobre programas en C y yo intentaré resolverlas. Para hacerlo deben rellenar el formulario que se encuentra en la página principal de mi web:

www.victorsanchez2.net

Si deseas obtener más ejemplos de C, Pascal, ensamblador, Visual Basic o JavaScript visita mi web.

Comencemos!!!!!!!!!! ;-)

El compilador más utilizado para compilar estos programas ha sido el TC++ 3.0 aunque también se ha utilizado en algunas ocasiones el gcc de Linux y el DevC++ para Windows.

**PRÁCTICA 1:**

El objetivo de esta práctica es afianzar ciertos conceptos básicos de la programación en lenguaje C:

- Bucles y tablas.
- Entrada y salida de datos.
- Paso de argumentos a un programa.
- Manejo de archivos.
- Funciones.
- Uso de memoria dinámica.

**EJERCICIOS**

**1.**

Escribir un programa (es decir una función *main*) que pida por teclado una serie de números enteros, los almacene en una tabla estática y posteriormente escriba por pantalla todos los números introducidos indicando además cual es el mayor y el menor.

Lo primero que debe hacer el programa es preguntar al usuario cuantos números se van a introducir y comprobar que dicha cantidad es menor que la dimensión de la tabla. Para dicha dimensión debe definirse una constante (por ejemplo MAX_DIMENSION) a la cual se puede asignar un valor razonable (por ejemplo, 100).

Funciones C a utilizar:

- *scanf*: para leer números introducidos por el teclado.
- *printf*: para imprimir por pantalla.

Ejemplo de ejecución: lo que se vería por pantalla debe ser algo así como

```
Introduce la cantidad de números:
3
Introduce los números:
5
7
2
Los números introducidos son:
5
7
2
El mayor es el 7.
El menor es el 2.
```

```
/*********************************************************
*   Autor: Victor Sanchez2
*   Web: www.victorsanchez2.net
*   Correo: victorss19@eresmas.com
*
*********************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define OK 1
#define ERR -10001
#define MAX_DIM 100

int enteros[MAX_DIM];         /* Array para guardar los enteros */
int num_total;


int lectura(void);
int sacarpantalla(int max, int min, int num_total);
int calcular(int *max, int *min, int total);
```

```
int es_digito(char caracter[]);


int main()
{
  int menor;              /* Numero menor */
  int mayor;              /* Numero mayor */

  if (lectura() == ERR)
   {
    printf("Error al leer el numero de terminos.\n");
    return ERR;
   }

   if (calcular(&mayor, &menor, num_total) == ERR)
     {
     printf("Error al intentar calcular el mayor y el menor");
     return ERR;
    }

   if (sacarpantalla(mayor, menor, num_total) == ERR)
    {
     printf("Error al mostrar por pantalla.\n");
     return ERR;
    }
 return OK;
}

/*****************************************************
*   Funcion: int lectura(int *total)
*
*   IN: Toma una variable tipo int.
*   OUT: Devuelve OK si se ha leido correctamente, ERR si hay algun fallo.
*   MAKE: Obtiene los numeros que introduce el usuario y los guarda
*         en un array de enteros
*****************************************************/
int lectura(void)
{
 int continuar = OK;
 int n;
 char numeros[MAX_DIM];

 for (n = 0; n < (int) strlen(numeros); n++)
  numeros[n] = '0';

 printf("\nCuantos numeros va a introducir? --> ");

 do {
  gets(numeros);
  continuar = OK;
  if ((num_total = es_digito(numeros)) == ERR)
    continuar = ERR;
  if (num_total <= 0)
   {
    printf("Debe introducir un numero positivo.\n");
    continuar = ERR;
   }
 } while (continuar != OK);

 printf("\nIntroduzca los numeros: \n");
 for (n = 0; n < num_total; n++)
  {
   gets(numeros);
   if ((enteros[n] = es_digito(numeros)) == ERR)
    {
     printf("Error al leer el numero, por favor repita introduccion.\n");
     n--;
    }
```

```
  }
 return OK;
}

/*******************************************************
*   Funcion: int calcular(int *max, int *min, int num_total)
*
*   IN: *max y *min que acumulan el numero mayor y menor. num_total
*       acumula la cantidad de numeros que se van a introducir.
*   OUT: OK si se ha realizado correctamente o ERR si hay fallos.
*   MAKE: Encuentra el minimo y el maximo de los numeros introducidos
*
*******************************************************/
int calcular(int *max, int *min, int total)
{
 int n;
 *min = *max = enteros[0];    /* Lo inicializamos */

 for (n = 1; n < total; n++)
  {
   if (*min > enteros[n])
       *min = enteros[n];
   if (*max < enteros[n])
       *max = enteros[n];
  }
 return OK;
}

/*******************************************************
*   Funcion: int sacarpantalla(int max, int min, int num_total)
*
*   IN: toma el valor maximo,minimo y total de elementos
*   OUT: devuelde ERR si no puede y OK si puede
*   MAKE: Nada
*
*******************************************************/
int sacarpantalla(int max, int min, int num_total)
{

 int n;

 printf("Los numeros introducidos son: ");

 for (n = 0; n < num_total; n++)
   printf("%d, ", enteros[n]);

 printf("\nMinimo: %d", min);
 printf("\nMaximo: %d\n", max);

 return OK;
}

/*******************************************************
*   Funcion: int es_digito(char caracter)
*
*   IN: Se le da una serie de caracteres para que se pasen a numero
*   OUT: Nos devolvera ERR si se produce un eroor
*   MAKE: Comprueba la integridad de los datos
*
*******************************************************/
int es_digito(char caracter[])
{

 int i;
 int num = 0;
 int rango;

 /* Si la cadena esta vacia */
```

```
 if (caracter[0] == 0)
   return ERR;

  if (caracter[0] == '-')     /* Si se ha introducido un numero negativo */
    {
     i = 1;
 /* Empezamos el bucle para ver si es caracter desde el siguiente caracter */
      rango = 5;
    }
  else              /* Debe ser un numero positivo o hay algun error */
    {
     i = 0;
     rango = 4;
    }

  /* Comprobamos que todos los elementos sean digitos */
  /* El mayor numero que permitimos es 9999 */

  if ((int) strlen(caracter) > rango)
   {
    printf("No puede introducir esa cantidad de numeros!!!\n");
    printf("Es demasiado grande. Introduzca numeros entre -9999 y 9999\n");
    return ERR;
   }

 for (i = i; i < (int) strlen(caracter); i++)
  if (caracter[i] < '0' || caracter[i] > '9')
   {
    printf("\nIntroduzca unicamente digitos, por favor.\n");
    return ERR;
   }


 /* Si solo se ha introducido un caracter */
 if (strlen(caracter) == 1)
  return num = caracter[0] - '0';  /* Devolvemos el digito */


 /* Si hay mas de un caracter */
 for (i = 0; i != (int) strlen(caracter); i++)
   num = (num * 10) + (caracter[i] - '0');

 return num;
}
```

**2.**

Escribir un programa (es decir una función *main*) que lea el contenido de un archivo de texto y lo vuelque por pantalla, indicando posteriormente la longitud de la línea más larga. El programa debe recibir el nombre del archivo como argumento, no teniendo que introducirse éste por pantalla. Para ello la función *main* debe tener el prototipo estándar **int main(int argc, char *argv[])**.

Funciones C a utilizar:

- *fopen*: para abrir un archivo.
- *fclose*: para cerrar un archivo.
- *fgets*: para leer una línea de un archivo.
- *strlen*: para obtener la longitud de una cadena.

```
/*****************************************************************************
*
*   Autor: Victor Sanchez2
*   Web: www.victorsanchez2.net
*   Correo: victorss19@eresmas.com
*   Make: Este algoritmo lee un archivo introducido por el usuario
*         y calcula cual es el numero de caracteres de la linea mayor.
*****************************************************************************/

#include <stdio.h>
#include <string.h>
#define ERR -1
#define OK 1
#define MAX_DIM 80       /* Maxima longitud de la linea */
#define mayor(a,b)((a) > (b) ? (a):(b))

char *fgets(char *linea, int max, FILE * fp);
int mayor_linea(void);
int leer_archivo(char *archivo[]);
void ayuda(void);

FILE * fp;
int max = MAX_DIM;

int main(int argc, char *argv[])
{
 /* Debe introducir <nombre del ejecutable> <nombre del archivo> */
 if (argc != 2)
  {
   ayuda();
   return ERR;
  }

 if (leer_archivo(&argv[1]) == ERR)
  return ERR;

 printf("\nLa linea mas larga tiene %d caracteres.\n", mayor_linea());

 fclose(fp);
 return OK;
}

/*****************************************************************************
*
*   Funcion: int leer_archivo(char *archivo[])
*
*   IN: Nombre del archivo
*   OUT: OK si se ha leido correctamente, ERR si ha habido algun fallo
*   MAKE: Lee un archivo introducido por el usuario.
*
*****************************************************************************/
int leer_archivo(char *archivo[])
{

 if ((fp = fopen(archivo[0], "r")) == NULL)
  {
   printf("Error al intentar abrir el archivo\n");
   return ERR;
  }

 return OK;
}

/*****************************************************************************
*
*   Funcion: int mayor_linea(void)
*
*   IN:
```

```
*   OUT:  Tamagno de la linea de mayor longitud del archivo.
*   MAKE: Calcula el tamagno de la linea mas larga.
*
*****************************************************************************/
int mayor_linea(void)
{

 char cad[MAX_DIM];
 int mayor_cad = 0;      /* Inicializamos el valor de la cadena */
 while (!feof(fp))
  {
   fgets(cad, MAX_DIM, fp);
   mayor_cad = mayor(mayor_cad, (int) strlen(cad));
  }

 return mayor_cad;
}

/*****************************************************************************
*
*   Funcion: void ayuda(void)
*
*   IN:
*   OUT: Mensaje de ayuda para el usuario.
*   MAKE: Muestra ayuda al usuario para que introduzca los datos
correctamente.
*
*****************************************************************************/
void ayuda(void)
{
  printf("\nDebe introducir: <nombre del ejecutable> <nombre del archivo>\n");
}
```

**3.**

El número ***e*** (2.7183...) se puede aproximar mediante la suma:

$$\sum_{n=0}^{N} \frac{1}{n!}$$

Sumatorio de 1/n! desde n=0 hasta n=N

En esta expresión la precisión queda determinada por el valor *N*. El valor *n!* representa el factorial de *n* (1·2·...·(*n* -1)·*n*). Recordemos que, por definición, 0!=1.

Escribir un programa (es decir una función *main*) que permita calcular la aproximación al número ***e*** dada por *N* (valor preguntado al usuario). Debe utilizarse una función auxiliar de prototipo **float Factorial (int n);** dicha función debe ser una función recursiva, puesto que n!=n·(n-1)!.

```
/*******************************************************
*   Autor: Victor Sanchez2
*   Web: www.victorsanchez2.net
*   Correo: victorss19@eresmas.com
*
*   MAKE: Este programa realiza la aproximacion al numero e.
```

```
*         Lo hemos realizado de 2 maneras:
*           - Recursivamente.
*           - Forma mas eficiente (sin usar recursividad).
*
*         Un ejemplo: (introduciendo 10000 terminos)
*           - Recursividad:  12 segundos aproximadamente
*           - Eficientemente: Se obtiene el resultado instantaneamente.
*
*********************************************************/

#include <stdio.h>
#include <string.h>
#define OK 1
#define ERR -10001
#define NO 0
#define MAX_TERM 1000000

float n_term;

int eficiente;
int lectura(int *eficiente, float *n_term);
float calcular(float n_term);
int sacarpantalla(float numeroe);
float factorial(int n);
int es_digito(char caracter[]);



int main()
{
 if (lectura(&eficiente, &n_term) == ERR)
  {
   printf("Error al leer el numero de terminos\n");
   return ERR;
  }

 if (sacarpantalla(calcular(n_term)) == ERR)
  {
   printf("Error al sacar por pantalla/calcular el numero\n");
   return ERR;
  }

 return OK;
}

/*******************************************************
*   Funcion: int sacarpantalla(float numeroe)
*
*   IN: Toma la variable numeroe de tipo float.
*   OUT: Devuelve OK si lo consigue, ERR si hay algun
*   MAKE: Saca por pantalla el resultado. No modifica ninguna variable.
*
*******************************************************/
int sacarpantalla(float numeroe)
{

 printf("\nEl valor de la aproximacion al numero e es: %f\n\n", numeroe);
 return OK;
}

/*******************************************************
*   Funcion: int lectura(int *eficiente, float *n_term)
*
*   IN: Toma la direccion de la variable eficiente, y el numero de términos
*   OUT: Nos devuelve OK si lo consigue ERR si hay algun fallo.
*   MAKE: Se encarga de leer el numero de terminos a calcular. Modifica
*         n_term y eficiente.
*
```

```
*******************************************************/
int lectura(int *eficiente, float *n_term)
{
 char numero[5];
 char temp = 0;
 int continuar = NO;

 do {
  printf("\nIntroduzca el numero de terminos a calcular del numero e: ");
  gets(numero);
  } while ((*n_term = es_digito(numero)) == ERR || (*n_term < 1));

 do {
   printf("Desea usar el modo de calculo rapido? [S,N]:\t");
   fflush(stdin);
   fflush(stdout);
   scanf("%c", &temp);
   fflush(stdin);
   if ((temp == 'S') || (temp == 'N') || (temp == 's') || (temp == 'n'))
    {
     continuar = OK;
     if ((temp == 'S') || (temp == 's'))
       *eficiente = OK;
     else
      {
       *eficiente = NO;
       if (strlen(numero) > 4) /* Permitimos hasta el numero 9999 */
        {
         printf("\nSi desea calcular este numero con recursividad puede
                  ocasionar problemas.");
         printf("\nIntentelo con la manera eficiente.\n");
         return ERR;
        }
      }
    }
   else
    {
     continuar = NO;
     printf("Por favor introduzca un valor valido!!! \n");
    }
  } while (continuar != OK);

 return OK;
}


/*******************************************************
*   Funcion: float calcular(float terminos)
*
*   IN: Numero de terminos para realizar la aproximacion.
*   OUT: La aproximacion al numero e si es correcto o ERR si ha habido un
*        fallo.
*   MAKE: Calcula la serie: 1 entre el factorial de un numero dado.
*
*******************************************************/

float calcular(float terminos)
{
 float e = 0;
 float facto = 1;        /* Va a ir acumulando el valor del factorial cuando
se haya
                            seleccionado "eficiente" */
 int n;

 if (eficiente == OK)
  {
   printf("\nSe esta calculando el numero e de la manera mas eficiente.");
   e = 1;
```

```
   for (n = 1; n < (int) terminos; ++n)
    {
     facto *= n;
     e += 1 / facto;
    }

   return e;
  }
 else   /* Calculamos la serie usando recursividad */
  {
   do {
     e += (1 / factorial((int) terminos--));
    } while (terminos >= 0);

   printf("\nSe esta calculando el numero e usando recursividad.");
  }

return e;
}

/*********************************************************
*   Funcion: float factorial (int n)
*
*   IN:  Numero de terminos de los que se calcularan sucesivos factoriales
*   OUT: El factorial del numero o ERR si ha habido algun error.
*   MAKE: Calcula el factorial de un numero dado.
*
*********************************************************/
float factorial(int n)
{
  if (n == 0)
    return 1;
  else
    return (n * factorial(n - 1));
}


/*********************************************************
*   Funcion: int es_digito(char caracter)
*
*   IN: Se le da una serie de caracteres para que se pasen a numero
*   OUT: Nos devolvera ERR si se produce un eroor
*   MAKE: Comprueba la integridad de los datos
*
*********************************************************/
int es_digito(char caracter[])
{

int i;
int num = 0;
int rango;

 /* Si la cadena esta vacía */
 if (caracter[0] == 0)
   return ERR;

 if (caracter[0] == '-') /* Si se ha introducido un numero negativo */
  {
   i = 1;
/* Empezamos el bucle para ver si es caracter desde el siguiente caracter */
   rango = 7;
  }
 else               /* Debe un numero positivo o hay algun error */
  {
   i = 0;
   rango = 7;
  }
```

```
  /* Comprobamos que todos los elementos sean digitos */
  /* El mayor numero permitido es 9999999 */

  if ((int) strlen(caracter) > rango)
    {
    printf("No puede introducir esa cantidad de numeros!!!\n");
    printf("Es demasiado grande. Rango: -9999999 y 9999999\n");
    return ERR;
   }

  for (i = i; i < (int) strlen(caracter); i++)
  if (caracter[i] < '0' || caracter[i] > '9')
   {
    printf("\nIntroduzca unicamente digitos, por favor.\n");
    return ERR;
   }

  /* Si solo se ha introducido un caracter */
  if (strlen(caracter) == 1)
    return num = caracter[0] - '0';     /* Devolvemos el digito */

  /* Si hay mas de un caracter */
  for (i = 0; i != (int) strlen(caracter); i++)
    num = (num * 10) + (caracter[i] - '0');

return num;
}
```

**4.**

Modificar el programa del ejercicio 1 para que en lugar de utilizar una tabla estática utilice una tabla alocada dinámicamente. La tabla debe alocarse una vez que el usuario ha indicado el número de elementos y debe liberarse al finalizar el programa.

Funciones C a utilizar:

- *malloc*: para alocar memoria dinámicamente.
- *free*: para liberar memoria previamente alocada.

```
/*****************************************************************************
*
*   Autor: Victor Sanchez2
*   Web: www.victorsanchez2.net
*   Correo: victorss19@eresmas.com
*
*****************************************************************************/


#include <stdio.h>
#include <alloc.h>
#include <stdlib.h>
#include <string.h>
#define OK 1
#define ERR -10001
#define NO 0
#define MAX_DIM 9999

int creartabla(int **ptabla, int *numero);
int rellenartabla(int *tabla, int numero);
```

```
int controldeerrores(int numero);
int calcular(int *tabla, int numero, int *max, int *min);
int mostrarvalores(int *tabla, int numero, int max, int min);
int es_digito(char caracter[]);
int *tabla = NULL;
int numero, max, min;

int main(void)
{
 if (creartabla(&tabla, &numero) == ERR)
   return ERR;

 if (rellenartabla(tabla, numero) == ERR)
   return ERR;

 calcular(tabla, numero, &max, &min);
 mostrarvalores(tabla, numero, max, min);

return OK;
}

/*************************************************************************
*
*   Funcion: int creartabla(int **ptabla, int *numero)
*
*   IN: la direccion de un puntero y la direccion del numero de elementos a
*       guardar
*   OUT: OK si se consigue, ERR si se falla
*   MAKE: Crea una tabla dinamica de enteros en donde se guardaran los numeros
*         introducidos
*
*************************************************************************/
int creartabla(int **ptabla, int *numero)
{

 int continuar;
 char elementos[MAX_DIM];

 do {
   continuar = OK;
   printf("Por favor introduzca el numero de elementos: ");
   gets(elementos);
   /* Si esta vacio, es negativo o 0... */
   if (elementos[0] == 0 || elementos[0] == '-' || elementos[0] == '0')
     continuar = ERR;
   if ((*numero = es_digito(elementos)) == ERR)
     continuar = ERR;
  } while (continuar != OK);

 *ptabla = (int *) malloc(*numero * sizeof(int));

 if (*ptabla == NULL)
     return ERR;

 return OK;
}

/*************************************************************************
*
*   Funcion: int rellenartabla(int *tabla, int numero)
*
*   IN: la direccion de la tabla, el numero de enteros a guardar
*   OUT: OK si se consigue ERR si se falla
*   MAKE: rellena la tabla con los numero que introduzca el usuario
*
*************************************************************************/
int rellenartabla(int *tabla, int numero)
{
```

```
 int i, continuar;
 char num[MAX_DIM];

 printf("Por favor introduzca los numeros: \n");

 for (i = 0; i < numero; ++i)
  {
   continuar = OK;
   gets(num);
   if (num[0] == 0)
    {
     printf("Deberia introducir algun digito.\n");
     continuar = ERR;
    }
   if ((*tabla++ = es_digito(num)) == ERR)
    {
     printf("Error al leer el numero.\n");
     continuar = ERR;
    }
   if (continuar == ERR)
      i--;
  }

return OK;
}

/*****************************************************************************
*
*   Funcion: int calcular(int *tabla, int numero, int *max, int *min)
*
*   IN: La direccion de la tabla, el numero de elementos, y la direccion del
*       numero mas grande y mas pequeño
*   OUT: OK si se consigue ERR si falla
*   MAKE: calcula el numero mas grande y el mas pequeño
*
*****************************************************************************/
int calcular(int *tabla, int numero, int *max, int *min)
{

 int i;
 *max = *min = tabla[1];

 for (i = 0; i < numero; i++)
  {
   if (tabla[i] > *max)
     *max = tabla[i];
   if (tabla[i] < *min)
     *min = tabla[i];
  }

return OK;
}

/*****************************************************************************
*
*   Funcion: int mostrarvalores(int *tabla, int numero, int max, int min)
*
*   IN: La direccion de la tabla y los valores del numero maximo, minimo y
*       numero de enteros
*   OUT: OK si se consigue ERR si falla
*   MAKE: mostrar por pantalla los valores
*
*****************************************************************************/
int mostrarvalores(int *tabla, int numero, int max, int min)
{

 int i;
```

```
 printf("Los valores introducidos han sido:\t");

 for (i = 0; i < numero; i++)
    printf("%d\t", tabla[i]);

 free(tabla);

 printf("\nEl valor maximo ha sido:\t%d", max);
 printf("\nEl valor minimo ha sido:\t%d", min);
 printf("\nEl numero de terminos introducidos ha sido: %d\t\n", numero);

 return OK;
}


/*********************************************************
*   Funcion: int es_digito(char caracter)
*
*   IN: una cadena de caracters
*   OUT: el valor del numero
*   MAKE: nada
*
*********************************************************/
int es_digito(char caracter[])
{

 int i;
 int num = 0;
 int rango;

 if (caracter[0] == '-') /* Si se ha introducido un numero negativo */
  {
   i = 1;
/* Empezamos el bucle para ver si es caracter desde el siguiente caracter */
   rango = 5;
  }
 else               /* Debe un numero positivo o hay algun error */
  {
   i = 0;
   rango = 4;
  }

 /* Comprobamos que todos los elementos sean digitos */
 if ((int) strlen(caracter) > rango)
/* El mayor numero que permitimos es 9999 */
  {
   printf("No puede introducir esa cantidad de numeros!!!\n");
   printf("Es demasiado grande. Introduzca numeros entre -9999 y 9999\n");
   return ERR;
  }

  for (i = i; i < (int) strlen(caracter); i++)
    if (caracter[i] < '0' || caracter[i] > '9')
     {
      printf("\nIntroduzca unicamente digitos, por favor.\n");
      return ERR;
     }

  /* Si solo se ha introducido un caracter */
  if (strlen(caracter) == 1)
    return num = caracter[0] - '0';     /* Devolvemos el digito */

  /* Si hay mas de un caracter */
  for (i = 0; i != (int) strlen(caracter); i++)
    num = (num * 10) + (caracter[i] - '0');

return num;
```

```
}
```

## PRÁCTICA 2:

OBJETIVOS

El objetivo de esta práctica es afianzar ciertos conceptos básicos de la programación en lenguaje C:
Estructuras y tipos de datos definidos.
Punteros.
Uso de memoria dinámica.
Comparaciones entre variables, cadenas, punteros.

### 1.

Definir un tipo de datos que represente a una persona, siendo sus campos: el nombre (char *), el apellido (char *), el nif (char [10]) y la edad (int). Llamar PERSONA a este tipo de datos. Escribir un programa que pida por pantalla esos datos y rellene una variable de tipo PERSONA. Prestar atención al hecho de que los miembros nombre y apellido deben alocarse dinámicamente, y liberarse posteriormente. Escribir una función auxiliar de prototipo void LiberarPersona(PERSONA); que libere dichos punteros.

- *Funciones C a utilizar:*
- *typedef : para definir un tipo de datos.*
- *gets: para leer una línea del teclado.*
- *malloc y free: para alocar y liberar memoria.*

```
/*****************************************************************************
*
*   Autor: Victor Sanchez2
*   Web: www.victorsanchez2.net
*   Correo: victorss19@eresmas.com
*   Make: Este programa pide los datos por pantalla de una estructura en la
*         que se guardaran: nombre, apellido, nif y edad. El nombre y el
*         apellido son alocados dinamicamente y posteriormente liberados.
*
*****************************************************************************/

#include <stdio.h>
#include <alloc.h>
#include <stdlib.h>
#include <string.h>
#define MAX_LONG_NIF 10    /* Longitud maxima del NIF */
#define MAXLEN 80          /* Longitud maxima de las cadenas que leemos */

typedef struct
 {
   char *nombre;
   char *apellido;
   char nif[MAX_LONG_NIF];
   int edad;
 } PERSONA;
```

```
PERSONA persona, *pPersona=NULL;
typedef enum { OK = 1, ERR = 0} status;


status LeerDatos(PERSONA *pp);
void   MostrarDatos(PERSONA *);
status EsDigito(char *caracter);
status CadenaVacia(char caracter[]);
int    PasaAEntero(char caracter[], int maximo);
status LiberarPersona(PERSONA *);

int main(void)
{
  pPersona = &persona;

  if (LeerDatos(pPersona) == ERR)
   {
    LiberarPersona(pPersona);
    return ERR;
   }

  MostrarDatos(pPersona);
  LiberarPersona(pPersona);

  return OK;
}

/*****************************************************************************
*
*   Funcion: status LeerDatos(PERSONA *pp)
*
*   IN: Puntero a una estructura de tipo PERSONA. Variable: *pp
*   OUT: OK si se leen los datos correctamente, ERR si hay algun fallo
*   MAKE: Pide los datos de la persona al usuario y lo guarda en la estructura
*
*****************************************************************************/
status LeerDatos(PERSONA *pp)
{
  #define LONG_NIF 8   /* Numero de digitos que debe tener el NIF */
  #define LONG_CADENA 60
  char temp[MAXLEN];
  int i;

  printf("Introduzca los datos de la persona:\n");

  printf("\nNombre: ");
  gets(temp);
  pp->nombre = (char *) malloc((strlen(temp) + 1) * sizeof(char));
  if (CadenaVacia(temp) == OK)
    return ERR;
  else if (strlen(temp) > LONG_CADENA)
   {
    printf("La longitud de la cadena es demasiado larga");
    return ERR;
   }
  else
    strcpy(pp->nombre, temp);

  fflush(stdout);
  printf("Apellido: ");
  gets(temp);
  pp->apellido = (char *) malloc((strlen(temp) + 1) * sizeof(char));
  if (CadenaVacia(temp) == OK)
    return ERR;
   else
     strcpy(pp->apellido, temp);
```

```
  for (i=0; i<strlen(pp->apellido); i++)
   {
    if (pp->apellido[i] == ' ')
     {
       printf("Introduzca un apellido unicamente y no deje espacios.");
      return ERR;
      }
   }

  printf("NIF (8 digitos, comience con 0 si es necesario): ");
  gets(temp);
  if (strlen(temp) != LONG_NIF)
   {
     printf("El NIF debe tener 8 digitos.\n");

     return ERR;
    }
   if (EsDigito(temp) == OK)
    {
     for (i=0; i<strlen(temp); i++)
       pp->nif[i] = temp[i];
    }
   else
     return ERR;

   printf("Edad: ");
   gets(temp);
   if (EsDigito(temp) == ERR)
      return ERR;
   pp->edad=PasaAEntero(temp, 2);
   if (pp->edad == ERR)
    {
     printf("\nError al intentar leer la edad.");
     printf("\nLa edad debe estar entre 1 y 99.\n");
     return ERR;
    }

  return OK;
}

/**************************************************************************
*
*   Funcion: void MostrarDatos(PERSONA *pp)
*
*   IN: Puntero a una estructura de tipo PERSONA
*   OUT:
*   MAKE: Muestra los datos de la estructura por pantalla
*
**************************************************************************/
void MostrarDatos(PERSONA *pp)
{
  printf("\nLos datos obtenidos son:\n");

  printf("\nNombre: %s", pp->nombre);
  printf("\nApellido: %s", pp->apellido);
  printf("\nNIF: %s", pp->nif);
  printf("\nEdad: %d", pp->edad);
}


/*****************************************************
*   Funcion: int EsDigito(char *caracter)
*
*   IN: Se le da una serie de caracteres para que se compruebe si es un numero
*   OUT: Nos devolvera ERR si se produce algun fallo, OK si ha ido todo bien
*   MAKE: Comprueba la integridad de los datos
*
*****************************************************/
```

```
status EsDigito(char *caracter)
{
 int i;

 CadenaVacia(caracter);

 for (i = 0; i < (int) strlen(caracter); i++)
  if (caracter[i] < '0' || caracter[i] > '9')
   {
    printf("\nIntroduzca unicamente digitos, por favor.\n");
    return ERR;
   }

 return OK;
}


/**************************************************************************
*
*   Funcion: status CadenaVacia(char caracter[])
*
*   IN: Una cadena de caracteres
*   OUT: ERR si la cadena no esta vacia, OK si la cadena esta vacia
*   MAKE: Comprueba si una cadena esta o no vacia
*
**************************************************************************/
status CadenaVacia(char caracter[])
{

  if (caracter[0] == 0)
   {
    printf("\nNo puede dejarlo en blanco, debe introducir los datos\n");
    return OK;
   }

  return ERR;
}


/**************************************************************************
*
*   Funcion: int PasaAEntero(char caracter[], int maximo)
*
*   IN: cadena de caracteres que tenemos que pasar a entero y maximo de
*       digitos permitidos.
*   OUT: El numero pasado a entero o ERR si hay algun fallo
*   MAKE: Pasa una cadena de caracteres a entero
*
**************************************************************************/
int PasaAEntero(char caracter[], int maximo)
{
 int num=0;
 int i;

 if (strlen(caracter) > maximo)
  {
   printf("\nDato erroneo.");
   return ERR;
  }

 /* Si solo se ha introducido un caracter */
 if (strlen(caracter) == 1)
   return num = caracter[0] - '0';  /* Devolvemos el digito */


 /* Si hay mas de un caracter */
 for (i = 0; i != (int) strlen(caracter); i++)
```

```
   num = (num * 10) + (caracter[i] - '0');  /* Lo devolvemos pasado a numero
*/

 return num;
}


/**************************************************************************
*
*   Funcion: status LiberarPersona(PERSONA *pp)
*
*   IN: Puntero a una estructura de tipo PERSONA
*   OUT: OK
*   MAKE: Libera la memoria que habiamos reservado previamente
*
**************************************************************************/
status LiberarPersona(PERSONA *pp)
{

 free(pp->nombre);
 free(pp->apellido);

 return OK;
}
```

## 2.

Escribir un programa (es decir una función *main*) que lea el contenido de un archivo de texto en el que cada línea contenga los datos que definen a una persona, y rellene de esa forma una tabla de variables de tipo PERSONA. El programa debe recibir el nombre del archivo como argumento, no teniendo que introducirse éste por pantalla. Para ello la función *main* debe tener el prototipo estándar **int main(int argc, char *argv[])**. El programa debe comprobar que el número de líneas que se van leyendo no supera la dimensión de la tabla definida. Una vez terminado de leer el archivo debe imprimirse por pantalla los datos de todas las personas (una por línea), y liberarse toda la memoria alocada dinámicamente.

Funciones C a utilizar:

- *fopen*: para abrir un archivo.
- *fclose*: para cerrar un archivo.
- *fgets*: para leer una línea de un archivo

```
/**************************************************************************
*
*   Autor: Victor Sanchez2
*   Web: www.victorsanchez2.net
*   Correo: victorss19@eresmas.com
*   Make: Este programa lee el contenido de un archivo de texto en el que cada
*         linea contiene los datos que definen a una persona y rellena una
*         tabla de tipo PERSONA. El nombre del archivo se recibe como
*         argumento. Cuando se ha terminado de leer el archivo se muestran por
*         pantalla los datos de las diferentes personas.
*
**************************************************************************/

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

```
#include <ctype.h>
#include <conio.h>

#define MAX_LEN 80
#define MAX_PERS 200
#define MAX_LONG_NIF 10

typedef struct
 {
   char *nombre;
   char *apellido;
   char nif[MAX_LONG_NIF];
   int edad;
 } PERSONA;

PERSONA persona[MAX_PERS], *pPersona=NULL;

typedef enum { OK = 1, ERR = 0} status;

void   ayuda(void);
status LeerArchivo(char *archivo[], PERSONA *pp, int *pTop);
status ControlErrores(char *pNombre, char *pApellido, char *pNif, char
*pEdad);
status CadenaVacia(char *caracter);
status EsDigito(char *caracter, int longitud);
int    PasaAEntero(char *caracter, int maximo, PERSONA *pDatos);
void   MostrarDatos(PERSONA *pp, int linea);
status LiberarPersona(PERSONA *);
status LiberarArray(PERSONA *pp, int *top);

int main(int argc, char *argv[])
{
 int top=0; /* Numero de personas leidas */

 clrscr();

 pPersona = persona; /* Puntero que apunta a una estructura de tipo PERSONA */

 /* Se debe introducir: <nombre del ejecutable> <nombre del archivo> */
 if (argc != 2)
  {
   ayuda();     /* Mensaje de ayuda para el usuario */
   return ERR;
  }

 if (LeerArchivo(&argv[1], pPersona, &top) == ERR)
  {
   printf("\nRevise el fichero, se produjo un fallo al intentar leerle.\n");
   return ERR;
  }

 return OK;
}

/*************************************************************************
*
*   Funcion: void ayuda(void)
*
*   IN:
*   OUT: Mensaje de ayuda para el usuario.
*   MAKE: Muestra ayuda al usuario para que introduzca los datos
*         correctamente.
*
*************************************************************************/
void ayuda(void)
{
  printf("Uso del programa: <nombre del ejecutable> <nombre del archivo>");
}
```

```
/*****************************************************************************
*
*   Funcion: status LeerArchivo(char *archivo[], PERSONA *pp, int *pTop)
*
*   IN: Nombre del archivo, puntero a una estructura de tipo PERSONA y maximo
*       numero de personas.
*   OUT: OK si se ha leido correctamente, ERR si ha habido algun fallo
*   MAKE: Lee un archivo introducido por el usuario.
*
*****************************************************************************/
status LeerArchivo(char *archivo[], PERSONA *pp, int *pTop)
{
 FILE *fp;
 int inicio=0;           /* Posicion en el array de estructuras */
 char cad[MAX_LEN];      /* Cadena con la que leemos las lineas del fichero */
 char nombre[MAX_LEN];
 char apellido[MAX_LEN];
 char nif[MAX_LEN];
 char edad[MAX_LEN];

 if (!(fp = fopen(archivo[0], "r")))
  {
   printf("\nError al intentar abrir el archivo\n");
   return ERR;
  }

 while (!feof(fp))
  {
   if (*pTop == MAX_PERS)
   /* Si se va a superar la dimension de la tabla definida*/
    {
      printf("\n\nSe ha sobrepasado el limite de PERSONAS.\n");
      return ERR;
     }
   pp=&persona[inicio];

   fgets(cad, MAX_LEN, fp);

   if (cad[0]=='\n')
    {
      printf("\n\nNo puede dejar lineas en blanco.");
     return ERR;
    }

   sscanf(cad, "%s %s %s %s", nombre, apellido, nif, edad);

   if(ControlErrores(nombre, apellido, nif, edad) == ERR)
     {
      printf("\nHa ocurrido un error al leer los datos.\n");
      return ERR;
     }

   /* Reservamos memoria y asignamos los valores */
   pp->nombre = (char *) malloc((strlen(nombre) + 1) * sizeof(char));
   strcpy(pp->nombre, nombre);
   pp->apellido = (char *) malloc((strlen(apellido) + 1) * sizeof(char));
   strcpy(pp->apellido, apellido);
   strcpy(pp->nif, nif);

   if (!feof(fp)) /* Si no hemos llegado al final del archivo... */
    {
     pp->edad = PasaAEntero(edad, 2, pp);
     if (pp->edad == ERR)
      return ERR;

     MostrarDatos(pp, inicio);
     inicio++;   /* Pasamos a la siguiente estructura de tipo PERSONA */
```

```
     (*pTop)++;
    }
  }

 if (LiberarArray(pp, pTop) == ERR)
   return ERR;
 fclose(fp);
 return OK;
}

/*************************************************************************
*
*   Funcion: status ControlErrores(char *pNombre, char *pApellido, char *pNif,
*                                  char *pEdad)
*
*   IN: Nombre, Apellido, NIF y Edad como cadena de caracteres
*   OUT: OK si las cadenas no contienen fallos, ERR si existe algun error
*   MAKE: Chequea que las cadenas no contengan datos no permitidos.
*
*************************************************************************/
status ControlErrores(char *pNombre, char *pApellido, char *pNif, char *pEdad)
{
 #define LONG_NIF 8
 /*Unico valor permitido para el numero de digitos del NIF */
 #define MAX_LONG_DATOS 17   /* Longitud maxima del nombre y del apellido */
 int i;

 if (strlen(pNombre) > MAX_LONG_DATOS)
  {
   printf("\n\nLa longitud del nombre que se estaba leyendo es demasidado
grande.");
   return ERR;
  }

 if (strlen(pApellido) > MAX_LONG_DATOS)
  {
   printf("\n\nLa longitud del apellido que se estaba leyendo es demasidado
              grande.");
   return ERR;
  }

 if (strlen(pNif) != LONG_NIF)
  {
   printf("\n\nEl NIF debe tener 8 digitos.");
   return ERR;
  }

 if(EsDigito(pNif, (int) strlen(pNif)) == ERR)
  {
   printf("\n\nError al leer el NIF de %s %s.", pNombre, pApellido);
   return ERR;
  }

 if (*pEdad == '0')
  {
   printf("\n\nSe ha leido una edad incorrecta. No puede tener 0 años.");
   return ERR;
  }

 for (i=0; i<strlen(pEdad); i++)
  {
   if (!isdigit(pEdad[i]))
    {
      printf("\n\nSe realizo una operacion no valida mientras se leia el
                  archivo.\n");
      return ERR;
    }
  }
```

```
 return OK;
}

/*****************************************************************************
*
*   Funcion: status CadenaVacia(char *caracter)
*
*   IN: Una cadena de caracteres
*   OUT: OK si la cadena esta vacia, ERR si la cadena contiene algun elemento
*   MAKE: Comprueba si la cadena esta vacia
*
*****************************************************************************/
status CadenaVacia(char *caracter)
{
  if (caracter[0] == 0)
   {
    printf("\nNo puede dejarlo en blanco, debe introducir los datos");
    return OK;
   }

  return ERR;
}

/*******************************************************
*   Funcion: status EsDigito(char *caracter, int longitud)
*
*   IN: Se le da una serie de caracteres para que compruebe si es un numero
*   OUT: Nos devolvera ERR si se produce algun fallo, OK si es correcto
*   MAKE: Comprueba la integridad de los datos
*
*******************************************************/
status EsDigito(char *caracter, int longitud)
{
 int i=0;

 if (CadenaVacia(caracter) == OK)
   return ERR;

 /* Comprobamos que todos los elementos sean digitos */
 for (i = i; i < longitud; i++)
  if (caracter[i] < '0' || caracter[i] > '9')
   {
    printf("\n\nIntroduzca unicamente digitos, por favor.");
    return ERR;
   }

 return OK;
}


/*****************************************************************************
*
*   Funcion: int PasaAEntero(char *caracter, int maximo)
*
*   IN: Cadena de caracteres que tenemos que pasar a entero y maximo de
*       digitos permitidos.
*   OUT: El numero pasado a entero o ERR si hay algun fallo.
*   MAKE: Pasa una cadena de caracteres a entero.
*
*****************************************************************************/
int PasaAEntero(char *caracter, int maximo, PERSONA *pDatos)
{
 int num=0;
 int i;

 if (strlen(caracter) > maximo)
  {
```

```
   printf("\n\n%s %s es muy mayor!!!", pDatos->nombre, pDatos->apellido);
   return ERR;
  }

 /* Si solo se ha introducido un caracter */
 if (strlen(caracter) == 1)
   return num = caracter[0] - '0'; /*Devolvemos el digito */

 /* Si hay mas de un caracter */
 for (i = 0; i != (int) strlen(caracter); i++)
   num = (num * 10) + (caracter[i] - '0');

 return num;
}

/*****************************************************************************
*
*   Funcion: void MostrarDatos(PERSONA *pp, int linea)
*
*   IN: Puntero a una estructura de tipo PERSONA y la linea en que debemos
*       ir escribiendo.
*   OUT:
*   MAKE: Muestra por pantalla los datos de la estructura.
*
*****************************************************************************/
void MostrarDatos(PERSONA *pp, int linea)
{

  gotoxy( 3,linea+1); printf("USUARIO: %s", pp->nombre);
  gotoxy(32,linea+1); printf("%s", pp->apellido);
  gotoxy(52,linea+1); printf("NIF: %s", pp->nif);
  gotoxy(70,linea+1); printf("EDAD: %d", pp->edad);

}

/*****************************************************************************
*
*   Funcion: int LiberarPersona(PERSONA *pp)
*
*   IN: Puntero a una estructura de tipo PERSONA.
*   OUT: OK si se liberan correctamente, ERR si hay algun error.
*   MAKE: Libera la memoria que reservamos anteriormente.
*
*****************************************************************************/
status LiberarPersona(PERSONA *pp)
{
 if(!pp->nombre || !pp->apellido) /* Si alguno es un puntero nulo */
  {
   printf("\n\nError al liberar el puntero");
   return ERR;
  }
 else
  {
   free(pp->nombre);
   free(pp->apellido);
  }

 return OK;
}

/*****************************************************************************
*
*   Funcion: int LiberarArray(PERSONA *pp, int *pTop)
*
*   IN: Puntero a una estructura de tipo PERSONA y el numero de elementos.
*   OUT: OK
*   MAKE: Libera cada parte de la estructura que le pasamos.
*
```

```
*************************************************************************/
status LiberarArray(PERSONA *pp, int *pTop)
{
    int i;

    for (i = 0; i < *pTop; i++)
     {
      LiberarPersona(--pp);
     }
    return OK;
}
```

**3.**

Modificar la definición del tipo PERSONA de forma tal que se añada un campo que sea un puntero a otra persona, que llamaremos el "amigo". Observar que para eso debe utilizarse la sentencia **struct** y no el tipo definido. Modificar el programa del ejercicio 2 para que de manera aleatoria se asigne a cada una de las personas de la tabla un *amigo*, y que este se imprima también por pantalla (es decir al imprimir los datos de una persona se imprime también el nombre y apellido de su *amigo*). Una vez terminado este programa modificarlo para que antes de imprimir los datos por pantalla libere la memoria de la segunda mitad de las personas (por ejemplo, del 5 al 9 si fuesen 10 en total) e imprima solo los datos de la primera mitad (en el ejemplo sería del 0 al 4). Explicar lo que ocurre.

Funciones C a utilizar:
*rand*: para generación de números aleatorios.

```
/*************************************************************************
*
*   Autor: Victor Sanchez2
*   Web: www.victorsanchez2.net
*   Correo: victorss19@eresmas.com
*   Make: Programa que asigna de manera aleatoria un amigo a cada una de las
*         personas de la tabla y lo imprime por pantalla. Una vez relizado
*         libera la memoria de la segunda mitad de las personas e imprime
*         unicamente los datos de la primera mitad.
*
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>
#include <time.h>

#define MAX_LONG_NIF 10
#define MAX_LEN 256
#define MAX_PERS 200

struct person
 {
   char *nombre;
   char *apellido;
   char nif[MAX_LONG_NIF];
   int edad;
   struct person *amigo;
 };
typedef struct person PERSONA;
PERSONA persona[MAX_PERS], *pPersona;

int top;
```

```
typedef enum { OK = 1, ERR = 0} status;

void   ayuda(void);
status LeerArchivo(char *archivo[], PERSONA *pp, int *pTop);
status AsignarAmigos(PERSONA *pp, int *top);
status ControlErrores(char *pNombre, char *pApellido, char *pNif, char
                      *pEdad);
status CadenaVacia(char *caracter);
status EsDigito(char *caracter, int longitud);
int    PasaAEntero(char *caracter, int maximo, PERSONA *pDatos);
void   MostrarPersona(PERSONA *pp, int *linea);
void   MostrarAmigo(PERSONA *pp, int *linea);
status AsesinarPersonas(PERSONA *pp, int *top);
status LiberarPersona(PERSONA *);

int main(int argc, char *argv[])
{
 int top = 0; /* Numero de personas en el archivo */
 int line1 = 1;
 int line2 = 2;
 int i;

 clrscr();

 pPersona = persona;

 /* Se debe introducir: <nombre del ejecutable> <nombre del archivo> */
 if (argc != 2)
  {
   ayuda();     /* Mensaje de ayuda para el usuario */
   return ERR;
  }

 if (LeerArchivo(&argv[1], pPersona, &top) == ERR)
  {
   printf("\nRevise el fichero, se produjo un fallo al intentar leerle.\n");
   return ERR;
  }

 if (AsignarAmigos(persona, &top) != OK)
   return ERR;

 pPersona = &persona[top/2];

 if (AsesinarPersonas(pPersona, &top) == ERR)
  {
   printf("\nError al asesinar a las personas.");
   return ERR;
  }
  printf("\n\nLa mitad de las personas han sido liberadas, pulse para
             continuar.\n");
  getch();
  clrscr();

  for (i=0; i<top; i++)
   {
    pPersona = &persona[i];
    MostrarPersona(pPersona, &line1);
    MostrarAmigo(pPersona, &line2);
   }

 return OK;
}

/*****************************************************************************
*
*   Funcion: void ayuda()
*
```

```
*   IN:
*   OUT: Mensaje de ayuda para el usuario.
*   MAKE: Muestra ayuda para que se introduzcna los datos correctamente.
*
*****************************************************************************/
void ayuda()
{
  printf("Uso: <nombre del ejecutable> <nombre del archivo de texto>\n");
}


/*****************************************************************************
*
*   Funcion: int LeerArchivo(char *archivo[], PERSONA *pp, int *pTop)
*
*   IN: Nombre del archivo, puntero a una estructura de tipo PERSONA y numero
*       de personas que hay.
*   OUT: OK si se ha leido correctamente, ERR si ha habido algun fallo.
*   MAKE: Lee un archivo introducido por el usuario y guarda los datos en una
*         estructura de tipo PERSONA.
*
*****************************************************************************/
status LeerArchivo(char *archivo[], PERSONA *pp, int *pTop)
{
 FILE *fp;
 int inicio=0;            /* Posicion en el array de estructuras */
 int linea = 1;           /* Nos va a servir para movernos por la pantalla */
 char cad[MAX_LEN];
 char nombre[MAX_LEN];
 char apellido[MAX_LEN];
 char nif[MAX_LEN];
 char edad[MAX_LEN];

 if (!(fp = fopen(archivo[0], "r")))
  {
   printf("\nError al intentar abrir el archivo\n");
   return ERR;
  }

 while (!feof(fp))
  {
   if (*pTop == MAX_PERS)
   /* Si se va a superar la dimension de la tabla definida*/
    {
      printf("\n\nSe sobrepaso el limite de PERSONAS.\n");
      return ERR;
     }
   pp=&persona[inicio];

   fgets(cad, MAX_LEN, fp);

   if (cad[0]=='\n')
    {
     printf("\n\nNo puede dejar lineas en blanco.");
     return ERR;
    }

   sscanf(cad, "%s %s %s %s", nombre, apellido, nif, edad);

   if(ControlErrores(nombre, apellido, nif, edad) == ERR)
     {
      printf("\nHa ocurrido un error al leer los datos.\n");
      return ERR;
     }

   /* Reservamos memoria y asignamos los valores */
   pp->nombre = (char *) malloc((strlen(nombre) + 1) * sizeof(char));
   strcpy(pp->nombre, nombre);
```

```
   pp->apellido = (char *) malloc((strlen(apellido) + 1) * sizeof(char));
   strcpy(pp->apellido, apellido);
   strcpy(pp->nif, nif);

   if (!feof(fp)) /* Si no hemos llegado al final del archivo... */
    {
     pp->edad = PasaAEntero(edad, 2, pp);
     if (pp->edad == ERR)
      return ERR;

     MostrarPersona(pp, &linea);
    /* Pasaremos a la siguiente estructura de tipo PERSONA */
     inicio++;
     (*pTop)++;
    }
  }
 fclose(fp);
 return OK;
}

/*****************************************************************************
*
*   Funcion: status AsignarAmigos(PERSONA *pp, int *top)
*
*   IN: Puntero a una estructura y el numero de personas.
*   OUT: OK
*   MAKE: Asigna una estructura llamada "amigo" a cada persona.
*
*****************************************************************************/
status AsignarAmigos(PERSONA *pp, int *top)
{
  int i;
  int aleat; /* Numero aleatorio */
  int linea = 2;

  srand((unsigned int) time((time_t *)NULL));
  for (i = 0; i != *top; i++)
   {
    aleat = rand() % (*top);
    pp=&persona[i];

    if (i != aleat)   /* Una persona no puede ser su propio amigo*/
      {
      pp->amigo = &persona[aleat];
       MostrarAmigo(pp, &linea);
      }
    else
       i--;
   }

  return OK;
}

/*****************************************************************************
*
*   Funcion: status ControlErrores(char *pNombre, char *pApellido, char *pNif,
*                                  char *pEdad)
*
*   IN: Nombre, apellido, NIF y edad de la persona que hemos leido.
*   OUT: OK si los datos no contienen fallos, ERR si se han encontrado
*        errores.
*   MAKE: Chequea que los datos leidos no contengan errores.
*
*****************************************************************************/
status ControlErrores(char *pNombre, char *pApellido, char *pNif, char *pEdad)
{
 #define LONG_NIF 8
                /* Unico valor permitido para el numero de digitos del NIF */
```

```
 #define MAX_LONG_DATOS 17     /* Longitud maxima del nombre y del apellido */
 int i;

 if (strlen(pNombre) > MAX_LONG_DATOS)
  {
   printf("\n\nLa longitud del nombre que se estaba leyendo es demasidado
               grande.");
   return ERR;
  }

 if (strlen(pApellido) > MAX_LONG_DATOS)
  {
   printf("\n\nLa longitud del apellido que se estaba leyendo es demasidado
               grande.");
   return ERR;
  }

 if (strlen(pNif) != LONG_NIF)
  {
   printf("\n\nEl NIF debe tener 8 digitos.");
   return ERR;
  }

 if(EsDigito(pNif, (int) strlen(pNif)) == ERR)
  {
   printf("\n\nError al leer el NIF de %s %s.", pNombre, pApellido);
   return ERR;
  }

 if (*pEdad == '0')
  {
   printf("\n\nSe ha leido una edad incorrecta. No puede tener 0 años.");
   return ERR;
  }

 for (i=0; i<strlen(pEdad); i++)
  {
   if (!isdigit(pEdad[i]))
    {
      printf("\n\nSe realizo una operacion no valida mientras se leia el
                  archivo.\n");
      return ERR;
     }
  }

 return OK;
}

/**************************************************************************
*
*   Funcion: int CadenaVacia(char *caracter)
*
*   IN: Una cadena de caracteres.
*   OUT: OK si la cadena esta vacia, ERR si la cadena contiene algun elemento.
*   MAKE: Chequea si la cadena introducida esta vacia.
*
**************************************************************************/
status CadenaVacia(char *caracter)
{
  if (caracter[0] == 0)
   {
    printf("\nNo puede dejarlo en blanco, debe introducir los datos");
    return OK;
   }

  return ERR;
}
```

```
/*******************************************************
*   Funcion: status EsDigito(char *caracter, int longitud)
*
*   IN: Se le da una serie de caracteres para que se compruebe si es un
*       numero.
*   OUT: Nos devolvera ERR si se produce algun fallo, OK si es correcto.
*   MAKE: Comprueba la integridad de los datos.
*
*******************************************************/
status EsDigito(char *caracter, int longitud)
{
 int i=0;

 if (CadenaVacia(caracter) == OK)
   return ERR;

 /* Comprobamos que todos los elementos sean digitos */
 for (i = i; i < longitud; i++)
  if (caracter[i] < '0' || caracter[i] > '9')
   {
    printf("\n\nIntroduzca unicamente digitos, por favor.");
    return ERR;
   }

 return OK;
}

/*****************************************************************************
*
*   Funcion: int PasaAEntero(char *caracter, int maximo, PERSONA *pDatos)
*
*   IN: Una cadena de caracteres, el maximo de digitos permitidos y un puntero
*       a una estructura de tipo PERSONA.
*   OUT: El numero convertido a entero o ERR si hay algun fallo.
*   MAKE: Pasa una cadena de caracteres a entero.
*
*****************************************************************************/
int PasaAEntero(char *caracter, int maximo, PERSONA *pDatos)
{
 int num=0;
 int i;

 if (strlen(caracter) > maximo)
  {
   printf("\n\n%s %s es muy mayor!!!", pDatos->nombre, pDatos->apellido);
   return ERR;
  }

 /* Si solo se ha introducido un caracter */
 if (strlen(caracter) == 1)
   return num = caracter[0] - '0'; /*Devolvemos el digito */

 /* Si hay mas de un caracter */
 for (i = 0; i != (int) strlen(caracter); i++)
   num = (num * 10) + (caracter[i] - '0');

 return num;
}

/*****************************************************************************
*
*   Funcion: void MostrarDatos(PERSONA *pp, int *linea)
*
*   IN: Un puntero a una estructura y la linea donde escribiremos en pantalla.
*   OUT:
*   MAKE: Muestra por pantalla los datos de la estructura.
*
*****************************************************************************/
```

```
void MostrarPersona(PERSONA *pp, int *linea)
{

  gotoxy( 3, *linea); printf("USUARIO: %s", pp->nombre);
  gotoxy(32, *linea); printf("%s", pp->apellido);
  gotoxy(52, *linea); printf("NIF: %s", pp->nif);
  gotoxy(70, *linea); printf("EDAD: %d", pp->edad);

  *linea += 2;
}

/*****************************************************************************
*
*   Funcion: void MostrarAmigo(PERSONA *pp, int *linea)
*
*   IN: Puntero a una estructura de tipo PERSONA y la linea en la que vamos
*       a escribir en la pantalla.
*   OUT:
*   MAKE: Muestra por pantalla los datos del amigo.
*
*****************************************************************************/
void MostrarAmigo(PERSONA *pp, int *linea)
{
  gotoxy( 3, *linea); printf("AMIGO:   %s", pp->amigo->nombre);
  gotoxy(32, *linea); printf("%s", pp->amigo->apellido);

  *linea += 2;
}


/*****************************************************************************
*
*   Funcion: int LiberarPersona(PERSONA *pp)
*
*   IN: Puntero a la estructura.
*   OUT: OK si todo es correcto, ERR si ha ocurrido algun error.
*   MAKE: Libera el nombre y apellido de la estructura introducida.
*
*****************************************************************************/
status LiberarPersona(PERSONA *pp)
{
 if(!pp->nombre || !pp->apellido) /* Si alguno es un puntero nulo */
  {
   printf("\n\nError al liberar el puntero");
   return ERR;
  }
 else
  {
   free(pp->nombre);
   free(pp->apellido);
  }

 return OK;
}

/*****************************************************************************
*
*   Funcion: status AsesinarPersonas(PERSONA *pp, int *top)
*
*   IN: Puntero a una estructura y el numero de personas.
*   OUT: OK
*   MAKE: Libera las personas que estan desde la mitad hasta el final.
*
*****************************************************************************/
status AsesinarPersonas(PERSONA *pp, int *top)
{
    int i;
```

```
    for (i = (*top/2); i < *top; i++)
       LiberarPersona(pp++);

    *top=((int)*top)/2;

    return OK;
}
```

**4.**

Modificar el programa del ejercicio 3 para que una vez que ha leído todos los datos, busque mediante un bucle si hay dos personas que tengan la misma edad. Repetir lo mismo para buscar si hay dos personas con el mismo nombre, apellido o nif. Repetir lo mismo para buscar si hay dos personas que tienen el mismo *amigo*.

Funciones C a utilizar:

strcmp: para comparar dos cadenas de caracteres.

```
/*****************************************************************************
*
*   Autor: Victor Sanchez2
*   Web: www.victorsanchez2.net
*   Correo: victorss19@eresmas.com
*   Make: Programa que asigna de manera aleatoria un amigo a cada una de las
*         personas de la tabla y lo imprime por pantalla. Una vez relizado
*         esto libera la memoria de la segunda mitad de las personas e imprime
*         unicamente los datos de la primera mitad.
*
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>
#include <time.h>

#define MAX_LONG_NIF 10
#define MAX_LEN 256
#define MAX_PERS 200

struct person
 {
   char *nombre;
   char *apellido;
   char nif[MAX_LONG_NIF];
   int edad;
   struct person *amigo;
 };
typedef struct person PERSONA;
PERSONA persona[MAX_PERS], *pPersona;

int top;
typedef enum { OK = 1, ERR = 0} status;

void   ayuda(void);
status LeerArchivo(char *archivo[], PERSONA *pp, int *pTop);
status AsignarAmigos(PERSONA *pp, int *top);
status ControlErrores(char *pNombre, char *pApellido, char *pNif, char
                      *pEdad);
status CadenaVacia(char *caracter);
status EsDigito(char *caracter, int longitud);
int     PasaAEntero(char *caracter, int maximo, PERSONA *pDatos);
```

```
void   MostrarPersona(PERSONA *pp, int *linea);
void   MostrarAmigo(PERSONA *pp, int *linea);
status LiberarArray(PERSONA *pp, int top);
status LiberarPersona(PERSONA *);
status CompararPersonas(PERSONA p[MAX_PERS], int top);

int main(int argc, char *argv[])
{
 int top = 0; /* Numero de personas en el archivo */
 int line1 = 1;
 int line2 = 2;
 int i;

 clrscr();

 pPersona = persona;

 /* Se debe introducir: <nombre del ejecutable> <nombre del archivo> */
 if (argc != 2)
  {
   ayuda();     /* Mensaje de ayuda para el usuario */
   return ERR;
  }

 if (!LeerArchivo(&argv[1], pPersona, &top))
  {
   printf("\nRevise el fichero, se produjo un fallo al intentar leerle.\n");
   return ERR;
  }

 if (AsignarAmigos(persona, &top) != OK)
   return ERR;

 for (i=0; i<top; i++)
  {
   pPersona = &persona[i];
   MostrarPersona(pPersona, &line1);
   MostrarAmigo(pPersona, &line2);
  }

 if (!CompararPersonas(persona, top))
   return ERR;

 pPersona = &persona[top];

 if (LiberarArray(pPersona, top) != OK)
   return ERR;


 return OK;
}

/**************************************************************************
*
*   Funcion: void ayuda()
*
*   IN:
*   OUT: Mensaje de ayuda para el usuario.
*   MAKE: Muestra ayuda al usuario para que introduzca los datos
*         correctamente.
*
**************************************************************************/
void ayuda()
{
  printf("Uso: <nombre del ejecutable> <nombre del archivo de texto>\n");
}
```

```
/*****************************************************************************
*
*   Funcion: int LeerArchivo(char *archivo[], PERSONA *pp, int *pTop)
*
*   IN: Nombre del archivo, puntero a una estructura de tipo PERSONA y numero
*       de personas que hay.
*   OUT: OK si se ha leido correctamente, ERR si ha habido algun fallo.
*   MAKE: Lee un archivo introducido por el usuario y guarda los datos en una
*         estructura de tipo PERSONA.
*
*****************************************************************************/
status LeerArchivo(char *archivo[], PERSONA *pp, int *pTop)
{
 FILE *fp;
 int inicio=0;            /* Posicion en el array de estructuras */
 int linea = 1;           /* Nos va a servir para movernos por la pantalla */
 char cad[MAX_LEN];
 char nombre[MAX_LEN];
 char apellido[MAX_LEN];
 char nif[MAX_LEN];
 char edad[MAX_LEN];

 if (!(fp = fopen(archivo[0], "r")))
  {
   printf("\nError al intentar abrir el archivo\n");
   return ERR;
  }

 while (!feof(fp))
  {
   if (*pTop == MAX_PERS)
   /* Si se va a superar la dimension de la tabla definida*/
    {
     printf("\n\nSe sobrepaso el limite de PERSONAS.\n");
     return ERR;
    }
   pp=&persona[inicio];

   fgets(cad, MAX_LEN, fp);

   if (cad[0]=='\n')
    {
     printf("\n\nNo puede dejar lineas en blanco.");
     return ERR;
    }

   sscanf(cad, "%s %s %s %s", nombre, apellido, nif, edad);

   if(!ControlErrores(nombre, apellido, nif, edad))
     {
      printf("\nHa ocurrido un error al leer los datos.\n");
      return ERR;
     }

   /* Reservamos memoria y asignamos los valores */
   pp->nombre = (char *) malloc((strlen(nombre) + 1) * sizeof(char));
   strcpy(pp->nombre, nombre);
   pp->apellido = (char *) malloc((strlen(apellido) + 1) * sizeof(char));
   strcpy(pp->apellido, apellido);
   strcpy(pp->nif, nif);

   if (!feof(fp)) /* Si no hemos llegado al final del archivo... */
    {
     pp->edad = PasaAEntero(edad, 2, pp);
     if (pp->edad == ERR)
      return ERR;

     MostrarPersona(pp, &linea);
```

```
    /* Pasaremos a la siguiente estructura de tipo PERSONA */
     inicio++;
     (*pTop)++;
    }
  }
 fclose(fp);
 return OK;
}

/******************************************************************************
*
*   Funcion: status AsignarAmigos(PERSONA *pp, int *top)
*
*   IN: Puntero a una estructura y el numero de personas.
*   OUT: OK
*   MAKE: Asigna una estructura llamada "amigo" a cada persona.
*
******************************************************************************/
status AsignarAmigos(PERSONA *pp, int *top)
{
  int i;
  int aleat; /* Numero aleatorio */
  int linea = 2;

  srand((unsigned int) time((time_t *)NULL));
  for (i = 0; i != *top; i++)
   {
    aleat = rand() % (*top);
    pp=&persona[i];

    if (i != aleat)   /* Una persona no puede ser su propio amigo*/
      {
      pp->amigo = &persona[aleat];
       MostrarAmigo(pp, &linea);
      }
    else
       i--;
   }

  return OK;
}

/******************************************************************************
*
*   Funcion: status ControlErrores(char *pNombre, char *pApellido, char *pNif,
char *pEdad)
*
*   IN: Nombre, apellido, NIF y edad de la persona que hemos leido.
*   OUT: OK si los datos no contienen fallos, ERR si se han encontrado
errores.
*   MAKE: Chequea que los datos leidos no contengan errores.
*
******************************************************************************/
status ControlErrores(char *pNombre, char *pApellido, char *pNif, char *pEdad)
{
 #define LONG_NIF 8          /* Unico valor permitido para el numero de
digitos del NIF */
 #define MAX_LONG_DATOS 17   /* Longitud maxima del nombre y del apellido */
 int i;

 if (strlen(pNombre) > MAX_LONG_DATOS)
  {
   printf("\n\nLa longitud del nombre que se estaba leyendo es demasidado
grande.");
   return ERR;
  }

 if (strlen(pApellido) > MAX_LONG_DATOS)
```

```
  {
   printf("\n\nLa longitud del apellido que se estaba leyendo es demasidado
grande.");
   return ERR;
  }

 if (strlen(pNif) != LONG_NIF)
  {
   printf("\n\nEl NIF debe tener 8 digitos.");
   return ERR;
  }

 if(EsDigito(pNif, (int) strlen(pNif)) == ERR)
  {
   printf("\n\nError al leer el NIF de %s %s.", pNombre, pApellido);
   return ERR;
  }

 if (*pEdad == '0')
  {
   printf("\n\nSe ha leido una edad incorrecta. No puede tener 0 años.");
   return ERR;
  }

 for (i=0; i<strlen(pEdad); i++)
  {
   if (!isdigit(pEdad[i]))
    {
      printf("\n\nSe realizo una operacion no valida mientras se leia el
archivo.\n");
      return ERR;
     }
  }

 return OK;
}

/**************************************************************************
*
*   Funcion: int CadenaVacia(char *caracter)
*
*   IN: Una cadena de caracteres.
*   OUT: OK si la cadena esta vacia, ERR si la cadena contiene algun elemento.
*   MAKE: Chequea si la cadena introducida esta vacia.
*
**************************************************************************/
status CadenaVacia(char *caracter)
{
  if (caracter[0] == 0)
   {
    printf("\nNo puede dejarlo en blanco, debe introducir los datos");
    return OK;
   }

  return ERR;
}

/*****************************************************
*   Funcion: status EsDigito(char *caracter, int longitud)
*
*   IN: Se le da una serie de caracteres para que se compruebe si es un
*       numero.
*   OUT: Nos devolvera ERR si se produce algun fallo, OK si es correcto.
*   MAKE: Comprueba la integridad de los datos.
*
*****************************************************/
status EsDigito(char *caracter, int longitud)
{
```

```
 int i=0;

 if (CadenaVacia(caracter) == OK)
   return ERR;

 /* Comprobamos que todos los elementos sean digitos */
 for (i = i; i < longitud; i++)
  if (caracter[i] < '0' || caracter[i] > '9')
   {
    printf("\n\nIntroduzca unicamente digitos, por favor.");
    return ERR;
   }

 return OK;
}

/**************************************************************************
*
*   Funcion: int PasaAEntero(char *caracter, int maximo, PERSONA *pDatos)
*
*   IN: Una cadena de caracteres, el maximo de digitos permitidos y un puntero
*       a una estructura de tipo PERSONA.
*   OUT: El numero convertido a entero o ERR si hay algun fallo.
*   MAKE: Pasa una cadena de caracteres a entero.
*
**************************************************************************/
int PasaAEntero(char *caracter, int maximo, PERSONA *pDatos)
{
 int num=0;
 int i;

 if (strlen(caracter) > maximo)
  {
   printf("\n\n%s %s es muy mayor!!!", pDatos->nombre, pDatos->apellido);
   return ERR;
  }

 /* Si solo se ha introducido un caracter */
 if (strlen(caracter) == 1)
   return num = caracter[0] - '0'; /*Devolvemos el digito */

 /* Si hay mas de un caracter */
 for (i = 0; i != (int) strlen(caracter); i++)
   num = (num * 10) + (caracter[i] - '0');

 return num;
}

/**************************************************************************
*
*   Funcion: void MostrarDatos(PERSONA *pp, int *linea)
*
*   IN: Un puntero a una estructura y la linea donde escribiremos en pantalla.
*   OUT:
*   MAKE: Muestra por pantalla los datos de la estructura.
*
**************************************************************************/
void MostrarPersona(PERSONA *pp, int *linea)
{

  gotoxy( 3, *linea); printf("USUARIO: %s", pp->nombre);
  gotoxy(32, *linea); printf("%s", pp->apellido);
  gotoxy(52, *linea); printf("NIF: %s", pp->nif);
  gotoxy(70, *linea); printf("EDAD: %d", pp->edad);

  *linea += 2;
}
```

```
/*****************************************************************************
*
*   Funcion: void MostrarAmigo(PERSONA *pp, int *linea)
*
*   IN: Puntero a una estructura de tipo PERSONA y la linea en la que vamos
*       a escribir en la pantalla.
*   OUT:
*   MAKE: Muestra por pantalla los datos del amigo.
*
*****************************************************************************/
void MostrarAmigo(PERSONA *pp, int *linea)
{
  gotoxy( 3, *linea); printf("AMIGO:   %s", pp->amigo->nombre);
  gotoxy(32, *linea); printf("%s", pp->amigo->apellido);

  *linea += 2;
}


/*****************************************************************************
*
*   Funcion: int LiberarPersona(PERSONA *pp)
*
*   IN: Puntero a la estructura.
*   OUT: OK si todo es correcto, ERR si ha ocurrido algun error.
*   MAKE: Libera el nombre y apellido de la estructura introducida.
*
*****************************************************************************/
status LiberarPersona(PERSONA *pp)
{
 if(!pp->nombre || !pp->apellido) /* Si alguno es un puntero nulo */
  {
   printf("\n.");
   return ERR;
  }
 else
  {
   free(pp->nombre);
   free(pp->apellido);
  }

 return OK;
}


/*****************************************************************************
*
*   Funcion: int LiberarArray(PERSONA *pp)
*
*   IN:  Puntero a una estructura de tipo PERSONA.
*   OUT:  OK
*   MAKE: Libera el array de estructuras.
*
*****************************************************************************/
status LiberarArray(PERSONA *pp, int top)
{
    int i;

    for (i = 0; i < top; i++)
     {
      LiberarPersona(--pp);
     }
    return OK;
}

/*****************************************************************************
*
*   Funcion: status CompararPersonas(PERSONA p[MAX_PERS], int top)
```

```
*
*   IN: Estructura con un tamaño [MAX_PERS], y el numero de personas
*   OUT: OK
*   MAKE: Muestra por pantalla las similitudes entre las personas.
*
*****************************************************************************/
status CompararPersonas(PERSONA p[MAX_PERS], int top)
{
  int i, j;

  printf("\n\nVamos a ver las comparaciones. Pulse una tecla.");
  getch();
  clrscr();

  printf("\n\n\ ");
  for (i=0; i<top; i++) {
    for(j=i+1; j<top; j++) {
      if (i!=j) {
        if (p[i].edad == p[j].edad)
         {
          printf("\nMisma edad(%d): %s", p[i].edad, p[i].nombre);
          printf("\n               %s", p[j].nombre);
         }
        if (!strcmp(p[i].nombre, p[j].nombre))
         {
          printf("\nMismo nombre(%s): %s", p[i].nombre, p[i].apellido);
          printf("\n                  %s", p[j].apellido);
         }
        if (!strcmp(p[i].apellido, p[j].apellido))
         {
          printf("\nMismo apellido (%s): %s", p[i].apellido, p[i].nombre);
          printf("\n                     %s", p[j].nombre);
         }
        if (!strcmp(p[i].nif, p[j].nif))
         {
          printf("\nMismo NIF (%s): %s", p[i].nif, p[i].nombre);
          printf("\n               %s", p[j].nombre);
         }
        if (p[i].amigo == p[j].amigo)
         {
          printf("\nMismo Amigo: %s", p[i].amigo->nombre);
          printf("\n                 %s", p[i].nombre);
          printf("\n                 %s", p[j].nombre);
         }
       }
     }
   }

    printf("\n");
    return OK;
}
```

## PRÁCTICA 3

### OBJETIVOS

El objetivo de esta práctica es implementar el tipo abstracto de datos (TAD) PILA y realizar un ejercicio de aplicación de dicho tipo de datos. El ejercicio consistirá en programar la resolución de laberintos: dado que el algoritmo básico para resolver un laberinto consiste en ir probando movimientos y volver atrás cuando ya no hay salida desandando el último movimiento, se presta a la utilización de una pila ya que lo que

hay que hacer es ir almacenando los movimientos que se han efectuado e irlos recuperando en orden inverso, es decir siguiendo el orden LIFO (Last In First Out).
La implementación del tipo de dato será una implementación genérica, es decir no dependiente del tipo de elemento que se va a guardar en la pila. Se va a utilizar el tipo de pila estática. Esto quiere decir que el espacio que reserva el dato para almacenar los elementos es de un tamaño dado que no se modifica posteriormente. Notemos que esto no quiere decir que dicho espacio no pueda reservarse dinámicamente (mediante *malloc*), sino que es un tamaño que una vez alocado no se modifica (y la pila devuelve error si se intenta rebasar dicho tamaño).
Si bien se incluyen en la práctica un par de ejercicios auxiliares para comprobar la implementación del tipo de datos PILA, la calificación de la práctica estará basada principalmente en el ejercicio número 4.

**EJERCICIOS**

1. Implementar las funciones básicas del TAD PILA. Dichas funciones están definidas en el archivo **pila.h** que se adjunta, y son las siguientes:

> **`ESTADO InicializaPila(PILA *pila)`:** reserva memoria para el array de datos de la pila, devuelve error si no ha conseguido alocar memoria.
>
> **`BOOLEANO PilaVacia(PILA pila)`:** indica si la pila está o no vacía.
>
> **`BOOLEANO PilaLlena(PILA pila)`:** indica si la pila está o no llena.
>
> **`ESTADO Push(PILA *pila, TIPO_INFO_PILA dato)`:** introduce un elemento en la pila, devuelve error si no ha sido posible (la pila está llena).
>
> **`ESTADO Pop(PILA *pila, TIPO_INFO_PILA *dato)`:** saca un elemento de la pila, devuelve error si no ha sido posible (la pila está vacía).
>
> **`void LiberaPila(PILA *pila)`:** libera la memoria alocada para el array interno de datos de la pila.

En dicho archivo se definen tanto los tipos de datos PILA, TIPO_INFO_PILA, ESTADO y BOOLEANO. El tipo de datos TIPO_INFO_PILA se define inicialmente como char, es decir los elementos de la pila en esta primera implementación serán caracteres.
La implementación de las funciones del TAD PILA debe realizarse en un archivo de nombre **pila.c**.

2. Escribir un programa de prueba para comprobar el funcionamiento adecuado de la implementación del tipo de datos PILA. El programa debe recibir como argumento por la línea de comandos una cadena de caracteres e imprimir por pantalla dicha cadena al revés. Para ello debe introducir todos los caracteres en la pila y sacarlos a continuación (en orden inverso). El código fuente del programa principal deberá estar en un archivo diferente al de las funciones del TAD PILA, debiendo generarse un *proyecto* con el TurboC para generar un ejecutable a partir de dos archivos. Es importante también que la utilización de la pila se realice a través de las funciones definidas en el ejercicio 1, y no a través de acceso directo a los miembros de la estructura de datos.

3. Modificar el archivo **pila.h,** de forma que el TIPO_INFO_PILA sea ahora un char *; renombrar el archivo pila1.h. Modificar el programa del ejercicio 2 para que reciba una serie de palabras como argumentos y las imprima en orden inverso.

4. Programación del algoritmo de resolución de laberintos. Para ello se codificará un laberinto como una matriz en la que cada casilla puede estar vacía (' ') o ocupada por un muro ('X'); existe también una casilla de entrada ('E') y otra de salida ('S'). Los laberintos tendrán siempre forma rectangular, y se podrán escribir en archivos de texto, teniendo por lo tanto que escribirse una función para leer el archivo y volcarlo a una estructura de datos correspondiente. En dicho archivo el final de una línea se indicará con la letra 'F' (la cual no forma parte del laberinto) para de este modo evitar confusiones conespacios enblanco o saltos de línea. La definición de dicha estructura y de las funciones necesarias se encuentra en el archivo **laberinto.h**, y las reproducimos a continuación.

```
typedef enum { NINGUNO, ARRIBA, ABAJO, DERECHA, IZQUIERDA

} MOVIMIENTO;

#define LADO_MAX 100

typedef struct {

       char mapa[LADO_MAX][LADO_MAX];

       int ancho;

       int alto;

} LABERINTO; /* definición del tipo laberinto */

typedef struct {

       int x;

       int y;

} COORDENADA;
```

**ESTADO lee_laberinto(char *nombre_fichero, LABERINTO *laberinto, COORDENADA *posicion_entrada):** Se encarga de volcar los datos desde el fichero a la estructura laberinto. Además, devuelve las coordenadas x e y de la entrada al laberinto. Da ERROR si hay algún problema.

**void imprime_laberinto(LABERINTO *laberinto):** Imprime el laberinto por pantalla.

**BOOLEANO es_casilla_permitida(LABERINTO laberinto, COORDENADA posicion_actual):** dice si podemos ocupar esa casilla del laberinto o no; podemos ocupar las casillas que tienen un espacio vacío, 'E' o 'S'.

**BOOLEANO es_casilla_salida(LABERINTO laberinto, COORDENADA posicion_actual):** dice si esa casilla es la salida ( 'S') o no.

**MOVIMIENTO politica_movimiento(MOVIMIENTO ultimo_movimiento):** en base al último movimiento que se ha realizado, nos dice qué movimiento debemos de intentar ahora. Por ejemplo: ninguno->arriba, arriba->derecha, derecha->abajo, abajo->izquierda, izquierda->ninguno. Esta función debe implementarse con un *switch*.

**MOVIMIENTO movimiento_opuesto(MOVIMIENTO movimiento):** para un movimiento dado, nos dice su opuesto. Izquierda y derecha son opuestos, arriba y abajo son opuestos. El opuesto de ninguno es ninguno. Esta función se debe implementar con un *switch*.

**COORDENADA calcula_nueva_posicion(COORDENADA posicion_actual, MOVIMIENTO siguiente_movimiento):** Esta función nos dice que coordenada tendremos si realizamos el movimiento especificado desde la posicion actual. Esta función se debe implementar con un *switch*.

**void marca_laberinto(LABERINTO *laberinto, COORDENADA posicion_actual):** Esta función marca el laberinto, en la posicion en la que estemos, con un punto (' . ').

La implementación de estas funciones deberá realizarse en un archivo de nombre **laberinto.c**.
A continuación debe programarse el algoritmo de resolución de laberinto, siguiendo el pseudo-código que se describe a continuación. Pseudo-código como referencia para escribir el código correspondiente:

```
inicializar la pila de movimientos

ultimo_movimiento_explorado = NINGUNO

salida_imposible = NO

salida_encontrada = es_casilla_salida(laberinto, posicion_actual);

mientras que no se haya encontrado la salida ni salida_imposible sea SI,

{

      marcar la posición actual en el laberinto

      hallar siguiente movimiento según nuestra política y meterlo en la
variable siguiente_opcion_movimiento

      si la variable siguiente_opcion_movimiento no es NINGUNO,

      {

            ultimo_movimiento_explorado = siguiente_opcion_movimiento

            calcular nueva posicion y meterla en la variable
siguiente_posicion

            si la casilla correspondiente a la nueva posicion es una casilla
      permitida

            {
```

```
                    meter el movimiento que hemos hecho en la pila

                    ultimo_movimiento_explorado = NINGUNO

                    posicion_actual = siguiente_posicion

             }

      }

      en caso contrario

      {

             sacar un dato de la pila de movimientos (este dato contiene el
             movimiento que me ha llevado a la actual posicion) y meterlo en
             ultimo_movimiento_explorado

             si no he podido sacar nada de la pila porque no había nada

                    salida_imposible = SI

             en caso contrario

                    calcular la nueva posición deshaciendo el movimiento que me
ha llevado a la casilla actual (tendré que moverme en la dirección
opuesta),asignándoselo a la variable posicion_actual

      }

      chequear si nos encontramos en la casilla de salida, en ese caso hacer
SI la variable salida_encontrada

} (fin del mientras)
```

A modo de resumen, el programa realizado deberá:

- Recibir el nombre del archivo con el laberinto desde la línea de comandos, comprobando por lo tanto que recibe los argumentos adecuados.
- Volcar los datos del laberinto a la matriz "laberinto" (función lee_laberinto). Comprobar la existencia del archivo, devolviendo error en caso contrario. Controlar otros errores como el intento de lectura de laberintos demasiado grandes, laberintos en los que las líneas tienen diferente tamaño, laberintos en los que no haya salida, o no entrada, laberintos en los que haya más de una entrada, etc.
- Realizar una búsqueda de un posible camino desde la entrada (E) a la salida (S) del laberinto. Se deberán de controlar errores como la existencia de laberintos en los que no hay ningún camino posible desde la entrada a la salida, etc.
- Dar como salida la secuencia de movimientos necesaria para salir desde la entrada (E) a la salida (S) del laberinto.

**Ejercicio 2:**

Este ejercico está compuesto de 7 archivos:

- infopila.h

- algorit.h
- tipos.h
- pila.h
- pila.c
- algorit.c
- main.c

1.infopila.h

```
#ifndef _INFO_PILA_H
#define _INFO_PILA_H
/* Definición del tipo de dato a almacenar en la pila */

typedef char TIPO_INFO_PILA;

/* fin del archivo infopila.h */
#endif
```

2.algorit.h

```
#ifndef ALGORIT_H
#define ALGORIT_H

ESTADO Algoritmo(int argc, char *argv[]);


#endif
```

3.tipos.h

```
#ifndef _TIPOS_H
#define _TIPOS_H

typedef enum {
      ERROR, BIEN
} ESTADO;

typedef enum {
  SI, NO
} BOOLEANO;


/* fin del archivo tipos.h */
#endif
```

4.pila.h

```
#ifndef _PILA_H
#define _PILA_H
#include "tipos.h"
#include "infopila.h"

#define MAX_PILA  1000  /* tamaño máximo de la pila */


 /* definición del nuevo tipo llamado PILA */
typedef struct {
     TIPO_INFO_PILA *datos;
     int            tope;
} PILA;
```

```
/*    Esta función inicializa la pila como vacía y reserva memoria suficiente
para un máximo
     de MAX_PILA elementos en la pila */
ESTADO InicializaPila(PILA *pila);

/*   Esta función nos dice si la pila está vacía o no */
BOOLEANO PilaVacia(PILA pila);


/*   Esta función nos dice si la pila está llena o no */
BOOLEANO PilaLlena(PILA pila);


/*  Devuelve BIEN si todo ha ido correctamente, ERROR en caso contrario */
ESTADO Push(PILA *pila, TIPO_INFO_PILA dato);


/*  Devuelve BIEN si todo ha ido correctamente, ERROR en caso contrario */
ESTADO Pop(PILA *pila, TIPO_INFO_PILA *dato);


/*    Esta función libera la memoria que se había reservado para los datos de
la pila */
void LiberaPila(PILA *pila);


/* fin del archivo pila.h */
#endif
```

## 5.pila.c

```
#include "tipos.h"
#include "infopila.h"
#include "pila.h"
#include <alloc.h>
#include <stdlib.h>
#define MAX_PILA  1000


/*    Esta funcion inicializa la pila como vacia y reserva memoria suficiente
para un maximo
     de MAX_PILA elementos en la pila */
ESTADO InicializaPila(PILA *pila)
{
  if (!pila)
    return ERROR;
  if      (!(pila->datos=      (TIPO_INFO_PILA      *)      malloc(MAX_PILA      *
sizeof(TIPO_INFO_PILA))))
    return ERROR;
  pila->tope=-1;

 return BIEN;
}


/*   Esta funcion nos dice si la pila está vacía o no */
BOOLEANO PilaVacia(PILA pila)
{
  if (pila.tope==-1)
    return SI;

 return NO;
}


/*   Esta funcion nos dice si la pila está llena o no */
BOOLEANO PilaLlena(PILA pila)
```

```
{
  if (pila.tope==MAX_PILA)
    return SI;

 return NO;
}


/*  Devuelve BIEN si todo ha ido correctamente, ERROR en caso contrario */
ESTADO Push(PILA *pila, TIPO_INFO_PILA dato)
{
  if (!pila)
    return ERROR;
  if (PilaLlena(*pila)==SI)
    return ERROR;
  pila->tope++;
  pila->datos[pila->tope]=dato;

 return BIEN;
}


/*  Devuelve BIEN si todo ha ido correctamente, ERROR en caso contrario */
ESTADO Pop(PILA *pila, TIPO_INFO_PILA *dato)
{
  if (!pila || !dato)
    return ERROR;
  if (PilaVacia(*pila)==SI)
    return ERROR;
  *dato=pila->datos[pila->tope];
  pila->tope--;

 return BIEN;
}


/*   Esta funcion libera la memoria que se había reservado para los datos de
la pila */
void LiberaPila(PILA *pila)
{
  free(pila->datos);
}
```

## 6.algorit.c

```
#include "pila.h"
#include "tipos.h"
#include "algorit.h"
#include <string.h>
#include <stdio.h>

ESTADO Algoritmo(int argc, char* argv[])
{
 int i=0, j=0, longitud;
 PILA pila;
 TIPO_INFO_PILA dato;

   if (!argv)
     return ERROR;
   if (!InicializaPila(&pila))
    {
     fprintf(stderr,"\nError  al  inicializar  la  pila,  posiblemente,  no  hay
memoria suficiente\n");
       return ERROR;
    }
   /* Controlamos que se metan parametros al comando */
   if (argc<2)
```

```
    {
     fprintf(stderr,"\nUso del programa: %s <PARAMETROS>",argv[0]);
     return ERROR;
    }

   /*Metemos la informacion en la pila...*/
   for(i=1;i<argc;i++)
    {
     longitud=strlen(argv[i]);
     Push(&pila,' ');
     for(j=0;j<longitud;j++)
       if(!Push(&pila,argv[i][j]))
        {
         fprintf(stderr,"\nError al manejar la pila, memoria insuficiente\n");
         return ERROR;
        }
    }

   /* Imprimamos la pila */
   while(PilaVacia(pila)==NO)
    {
     Pop(&pila,&dato);
     fprintf(stdout,"%c",dato);
    }

   fprintf(stdout,"\n");

   /*Liberemos la pila */
   LiberaPila(&pila);

 return BIEN;
}
```

7.main.c

```
#include "pila.h"
#include "infopila.h"
#include "algorit.h"
#include <stdio.h>

int main(int argc, char *argv[])
{
  if (!Algoritmo(argc,argv))
   {
    fprintf(stderr,"\nError al realizar el algoritmo.\n");
    return ERROR;
   }

 return BIEN;
}
```

**Ejercicio 3:**

Este ejercico está compuesto de 7 archivos:

- infopila.h
- algorit.h
- tipos.h

- pila.h
- pila.c
- algorit.c
- main.c

1.infopila.h

```
#ifndef _INFO_PILA_H
#define _INFO_PILA_H
/* Definición del tipo de dato a almacenar en la pila */

typedef char * TIPO_INFO_PILA;

#endif
```

2.algorit.h

```
#ifndef ALGORIT_H
#define ALGORIT_H

ESTADO Algoritmo(int argc, char *argv[]);

#endif
```

3.tipos.h

```
#ifndef _TIPOS_H
#define _TIPOS_H

typedef enum {
  ERROR, BIEN
} ESTADO;

typedef enum {
  SI, NO
} BOOLEANO;

#endif
```

4.pila.h

```
#ifndef _PILA_H
#define _PILA_H

#include "tipos.h"
#include "infopila.h"

#define MAX_PILA  1000  /* tamaño maximo de la pila */


 /* definicion del nuevo tipo llamado PILA */
typedef struct {
     TIPO_INFO_PILA *datos;
     int             tope;
} PILA;

/*   Esta funcion inicializa la pila como vacia y reserva memoria suficiente
     para un maximo de MAX_PILA elementos en la pila */
ESTADO InicializaPila(PILA *pila);

/*   Esta funcion nos dice si la pila esta vacia o no */
BOOLEANO PilaVacia(PILA pila);
```

```
/*   Esta funcion nos dice si la pila esta llena o no */
BOOLEANO PilaLlena(PILA pila);


/*  Devuelve BIEN si todo ha ido correctamente, ERROR en caso contrario */
ESTADO Push(PILA *pila, TIPO_INFO_PILA dato);


/*  Devuelve BIEN si todo ha ido correctamente, ERROR en caso contrario */
ESTADO Pop(PILA *pila, TIPO_INFO_PILA *dato);


/*   Esta funcion libera la memoria que se habia reservado para los datos de
la pila */
void LiberaPila(PILA *pila);

#endif
```

## 5.pila.c

```
#include "tipos.h"
#include "infopila.h"
#include "pila.h"
#include <stdlib.h>
#include <alloc.h>
#define MAX_PILA  1000


/*   Esta funcion inicializa la pila como vacia y reserva memoria suficiente
para un maximo
     de MAX_PILA elementos en la pila */
ESTADO InicializaPila(PILA *pila)
{
  if (!pila)
    return ERROR;
  pila->datos= (TIPO_INFO_PILA *) malloc(MAX_PILA * sizeof(TIPO_INFO_PILA));
  if (!pila->datos)
    return ERROR;
  pila->tope=-1;

 return BIEN;
}


/*   Esta funcion nos dice si la pila esta vacia o no */
BOOLEANO PilaVacia(PILA pila)
{
  if (pila.tope==-1)
    return SI;

 return NO;
}


/*   Esta funcion nos dice si la pila esta llena o no */
BOOLEANO PilaLlena(PILA pila)
{
  if (pila.tope==MAX_PILA)
    return SI;

 return NO;
}


/*  Devuelve BIEN si todo ha ido correctamente, ERROR en caso contrario */
ESTADO Push(PILA *pila, TIPO_INFO_PILA dato)
```

```
{
  if (!pila)
    return ERROR;
  if (PilaLlena(*pila)==SI)
    return ERROR;
  pila->tope++;
  pila->datos[pila->tope]=dato;

 return BIEN;
}


/*  Devuelve BIEN si todo ha ido correctamente, ERROR en caso contrario */
ESTADO Pop(PILA *pila, TIPO_INFO_PILA *dato)
{
  if (!pila || !dato)
    return ERROR;
  if (PilaVacia(*pila)==SI)
    return ERROR;
  *dato=pila->datos[pila->tope];
  pila->tope--;

 return BIEN;
}


/*   Esta funcion libera la memoria que se habia reservado para los datos de
la pila */
void LiberaPila(PILA *pila)
{
  free(pila->datos);
}
```

## 6.algorit.c

```
#include "pila.h"
#include "tipos.h"
#include "algorit.h"
#include <string.h>
#include <stdio.h>

ESTADO Algoritmo(int argc, char* argv[])
{
 int i=0, longitud;
 PILA pila;
 TIPO_INFO_PILA dato;

  if (!argv) return ERROR;
  if (!InicializaPila(&pila))
   {
    fprintf(stderr,"\nError al inicializar la pila, posiblemente, no hay
memoria suficiente\n");
    return ERROR;
   }
  /* Controlamos que se metan parametros al comando */
  if (argc<2)
   {
    fprintf(stderr,"\nUso del programa: %s <PARÁMETROS>",argv[0]);
    return ERROR;
   }

  /*Metemos la informacion en la pila...*/
  for(i=1;i<argc;i++)
   {
    longitud=strlen(argv[i]);
    Push(&pila," ");
    if(!Push(&pila,argv[i]))
```

```
     {
      fprintf(stderr,"\nError al manejar la pila, memoria insuficiente\n");
      return ERROR;
     }
   }

  /* Imprimamos la pila */
  while(PilaVacia(pila)==NO)
   {
    Pop(&pila,&dato);
    fprintf(stdout,"%s",dato);
   }

  fprintf(stdout,"\n");

  /*Liberemos la pila */
  LiberaPila(&pila);

 return BIEN;
}
```

7.main.c

```
#include "pila.h"
#include "infopila.h"
#include "algorit.h"
#include <stdio.h>

int main(int argc, char *argv[])
{
  if (!Algoritmo(argc,argv))
   {
    fprintf(stderr,"\nError al realizar el algoritmo.\n");
    return ERROR;
   }

 return BIEN;
}
```

**Ejercicio 4:**

Este ejercico está compuesto de 9 archivos:

1. infopila.h
2. algorit.h
3. tipos.h
4. pila.h
5. lab.h
6. pila.c
7. lab.c
8. algorit.c
9. main.c

Todos los archivos son necesarios para que el programa funcione correctamente. Para que nuestro compilador lo entienda correctamente deberemos generar un proyecto en TC o el compilador que utilicemos.

1.infopila.h

```
#ifndef _INFO_PILA_H
```

```
#define _INFO_PILA_H

#include "tipos.h"
#include "lab.h"

/* Definicion del tipo de dato a almacenar en la pila */
typedef MOVIMIENTO TIPO_INFO_PILA;

#endif
```

## 2.algorit.h

```
#ifndef _ALGORIT_H
#define _ALGORIT_H

#include "tipos.h"
#include "lab.h"

/* Resuelve el laberinto */
ESTADO Algoritmo(LABERINTO *laberinto, COORDENADA coordenada, int parametro,
BOOLEANO *solucion);

#endif
```

## 3. tipos.h

```
#ifndef _TIPOS_H
#define _TIPOS_H

typedef enum {
      ERROR, BIEN, SALIDA_IMPOSIBLE, ERROR_COMANDO
} ESTADO;

typedef enum {
  SI, NO
} BOOLEANO;

#endif
```

## 4.pila.h

```
#ifndef _PILA_H
#define _PILA_H

#include "tipos.h"
#include "infopila.h"

#define MAX_PILA  1000  /* Maximo de elementos posibles en la pila */


 /* Definicion del nuevo tipo llamado PILA */
typedef struct {
     TIPO_INFO_PILA *datos;
     int            tope;
} PILA;


/* Esta funcion inicializa la pila como vacia y reserva memoria suficiente
   para un maximo de MAX_PILA elementos en la pila */
ESTADO InicializaPila(PILA *pila);

/* Esta funcion nos dice si la pila esta vacia o no */
BOOLEANO PilaVacia(PILA *pila);


/* Esta funcion nos dice si la pila esta llena o no */
```

```
BOOLEANO PilaLlena(PILA *pila);


/* Devuelve BIEN si todo ha ido correctamente, ERROR en caso contrario */
ESTADO Push(PILA *pila, TIPO_INFO_PILA dato);


/* Devuelve BIEN si todo ha ido correctamente, ERROR en caso contrario */
ESTADO Pop(PILA *pila, TIPO_INFO_PILA *dato);


/* Mostramos los movimientos que se han realizado en el laberinto */
ESTADO mostrar_movimientos(PILA *pPila);


/* Esta funcion libera la memoria que se habia reservado para los datos de la
pila */
void LiberaPila(PILA *pila);


#endif
```

## 5.lab.h

```
#ifndef _LAB_H
#define _LAB_H

#include "tipos.h"
#define LADO_MAX 100  /* Dimension maxima permitida para el laberinto */

typedef enum {
  NINGUNO, ARRIBA, ABAJO, DERECHA, IZQUIERDA
 } MOVIMIENTO;


typedef struct {

 char mapa[LADO_MAX][LADO_MAX];
 int ancho;
 int alto;

} LABERINTO;             /* Definimos el tipo del laberinto */


typedef struct {

 int x;
 int y;

} COORDENADA;



/* Se encarga de volcar los datos desde el fichero a la estructura laberinto.
Además, devuelve las coordenadas x e y de la entrada al laberinto. Da ERROR si
hay algún problema. */
ESTADO lee_laberinto(char *nombre_fichero, LABERINTO * laberinto, COORDENADA *
posicion_entrada);

/* Imprime el laberinto por pantalla */
void imprime_laberinto(LABERINTO * laberinto);

/* Dice si podemos ocupar esa casilla del laberinto o no; podemos ocupar las
casillas que tienen un espacio vacío, 'E' o 'S'.*/
BOOLEANO      es_casilla_permitida(LABERINTO      *laberinto,      COORDENADA
posicion_actual);
```

```
/* Dice si esa casilla es la salida ( 'S') o no. */
BOOLEANO es_casilla_salida(LABERINTO *laberinto, COORDENADA posicion_actual);

/* Comprobamos si la casilla de salida esta rodeada por X */
BOOLEANO salida_rodeada(LABERINTO *laberinto, COORDENADA coord_salida);

/* En base al ultimo movimiento que se ha realizado, nos dice que movimiento
debemos de intentar ahora. Por ejemplo: ninguno->arriba, arriba->derecha,
derecha->abajo, abajo->izquierda, izquierda->ninguno.*/
MOVIMIENTO politica_movimiento(MOVIMIENTO *ultimo_movimiento);

/* Para un movimiento dado, nos dice su opuesto. Izquierda y derecha son
opuestos, arriba y abajo son opuestos. El opuesto de ninguno es ninguno. */
MOVIMIENTO movimiento_opuesto(MOVIMIENTO movimiento);

/* Esta funcion nos dice que coordenada tendremos si realizamos el movimiento
especificado desde la posicion actual.*/
COORDENADA calcula_nueva_posicion(COORDENADA posicion_actual, MOVIMIENTO
siguiente_movimiento);

/* Esta funcion marca el laberinto, en la posicion en la que estemos, con un
punto (' . )*/
void marca_laberinto(LABERINTO * laberinto, COORDENADA posicion_actual);

/* Esta funcion marca el laberinto, en la posicion en la que estemos, con un
punto (' . )*/
void marca_laberintoMalCamino(LABERINTO * laberinto, COORDENADA
posicion_actual);

/* Esta funcion marca el laberinto, en la posicion en la que estemos, con una
E (' E )*/
void marca_con_E(LABERINTO * laberinto, COORDENADA posicion_actual);

/* Imprime la solucion del laberinto por pantalla */
void imprime_laberinto_solucion(LABERINTO * laberinto);

#endif
```

## 6.pila.c

```
#include "tipos.h"
#include "infopila.h"
#include "pila.h"
#include <alloc.h>
#include <stdlib.h>
#include <stdio.h>
#define MAX_PILA  1000  /* Elementos que puede contener la pila */


/*****************************************************************************
*
*   Funcion: ESTADO InicializaPila(PILA *pila)
*
*   IN:   Puntero a la pila.
*   OUT:  Error si no se ha podido inicializar la pila.
*     MAKE: Esta funcion inicializa la pila como vacia y reserva memoria
suficiente
*         para un maximo de MAX_PILA elementos en la pila.
*
*****************************************************************************/
ESTADO InicializaPila(PILA *pila)
{
  if (!pila)
    return ERROR;

  pila->datos= (TIPO_INFO_PILA *) malloc(MAX_PILA * sizeof(TIPO_INFO_PILA));
  if (!(pila->datos))
```

```
    return ERROR;

  pila->tope=-1;

 return BIEN;
}


/**************************************************************************
*
*   Funcion: BOOLEANO PilaVacia(PILA *pila)
*
*   IN: Puntero a la pila.
*   OUT: Devuelve SI, si la pila no contiene elementos.
*   MAKE: Esta funcion nos dice si la pila esta vacia o no.
*
**************************************************************************/
BOOLEANO PilaVacia(PILA *pila)
{
  if (pila->tope == -1)
    return SI;

 return NO;
}


/**************************************************************************
*
*   Funcion: BOOLEANO PilaLlena(PILA *pila)
*
*   IN: Puntero a la pila.
*   OUT: Devuelve SI si la pila esta llena.
*   MAKE: Esta funcion nos informa de si la pila esta llena o no
*
**************************************************************************/
BOOLEANO PilaLlena(PILA *pila)
{
  if (pila->tope == MAX_PILA)
    return SI;

  return NO;
}


/**************************************************************************
*
*   Funcion: ESTADO Push(PILA *pila, TIPO_INFO_PILA dato)
*
*   IN: Puntero a la pila y el dato que se introduce en la pila.
*   OUT: Devuelve BIEN si todo ha ido correctamente, ERROR en caso contrario.
*   MAKE: Introducimos un dato en la pila si es posible.
*
**************************************************************************/
ESTADO Push(PILA *pila, TIPO_INFO_PILA dato)
{
  if (!pila)
    return ERROR;
  if (PilaLlena(pila)==SI)
    return ERROR;
  pila->tope++;
  pila->datos[pila->tope]=dato;

 return BIEN;
}


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

```
*   Funcion: ESTADO Pop(PILA *pila, TIPO_INFO_PILA *dato)
*
*   IN: Puntero a la pila y puntero al tipo de dato.
*   OUT: Devuelve BIEN si todo ha ido correctamte, ERROR en caso contrario.
*   MAKE: Sacamos un dato de la pila si es posible.
*
*****************************************************************************/
ESTADO Pop(PILA *pila, TIPO_INFO_PILA *dato)
{
  if (!pila || !dato)
    return ERROR;
  if (PilaVacia(pila)==SI)
    return ERROR;
  *dato=pila->datos[pila->tope];
  pila->tope--;

 return BIEN;
}


/*****************************************************************************
*
*   Funcion: ESTADO mostrar_movimientos(PILA *pPila)
*
*   IN: Puntero a la pila.
*   OUT: Se muestran por pantalla los movimientos realizados.
*   MAKE: Mostramos los movimientos que se han realizado en el laberinto.
*
*****************************************************************************/
ESTADO mostrar_movimientos(PILA *pPila)
{
 PILA PilaAux;
 TIPO_INFO_PILA elemento;

 InicializaPila(&PilaAux);

 while (PilaVacia(pPila) == NO)
  {
   Pop(pPila, &elemento);
   Push(&PilaAux, elemento);
  }

 while (PilaVacia(&PilaAux) == NO)
  {
   Pop(&PilaAux, &elemento);
   switch (elemento)
    {
     case ARRIBA:     printf("Arriba->");    break;
     case DERECHA:    printf("Derecha->");   break;
     case ABAJO:      printf("Abajo->");     break;
     case IZQUIERDA:  printf("Izquierda->"); break;
    }
  }
 printf("\n");

 LiberaPila(&PilaAux);

 return BIEN;
}



/*****************************************************************************
*
*   Funcion: void LiberaPila(PILA *pila)
*
*   IN: Puntero a la pila.
*   OUT:
```

```
*   MAKE: Libera la memoria que se habia reservado para los datos de la pila.
*
*****************************************************************************/
void LiberaPila(PILA *pila)
{
  free(pila->datos);
}
```

## 7.lab.c

```
#include "lab.h"
#include "tipos.h"
#include <stdio.h>

/*****************************************************************************
*
*          Funcion:    ESTADO    lee_laberinto(char    *nombre_fichero,LABERINTO
*laberinto,COORDENADA *posicion_entrada)
*
*    IN: El nombre del fichero, un puntero al laberinto y un puntero a la
posicion de entrada.
*      OUT: Posicion de entrada al laberinto y la estructura laberinto
completada. ERROR si algo
*        ha ido mal.
*    MAKE: Se encarga de volcar los datos desde el fichero a la estructura
laberinto. Ademas,
*          devuelve las coordenadas x e y de la entrada al laberinto. Devuelve
error si hay
*         algun problema.
*
*****************************************************************************/
ESTADO lee_laberinto(char *nombre_fichero, LABERINTO * laberinto, COORDENADA *
posicion_entrada)
{
  FILE *ptr=NULL;
  char c;             /* Variable para ir leyendo los caracteres del archivo */
  BOOLEANO entrada = NO, salida = NO;   /* Para comprobar el numero de
                                           entradas y salidas      */
  BOOLEANO salida_imposible=NO;         /* Mostraremos si el laberinto no
                                           tiene solucion */
  COORDENADA coordenada_salida;
  int x = 0, y = 0;

  if (!laberinto || !nombre_fichero || !posicion_entrada)
     return ERROR;

  laberinto->ancho = -1;

  /* Abrimos el archivo para lectura */
  ptr = fopen(nombre_fichero, "r");
  if (!ptr)
   {
     fprintf(stderr, "\nNo se pudo leer el archivo dado, por favor revise
rutas y permisos.");
     return ERROR;
   }

  /* Mientras que no lleguemos al final del archivo vamos comprobando y
acumulando el laberinto */
  while ((c = fgetc(ptr)) != EOF)
   {
    if (x < 0)
      {
       if (c == '\n')
        {
         x = -1;
```

```
          if (y==LADO_MAX)
           {
            fprintf(stderr,"\nLaberinto demasiado alto, por favor reviselo\n");
            return ERROR;
           }
          y++;
         }
        else
           x--;
       }
      else
       {
        switch (c)
         {
          case 'e':
          case 'E': if (entrada == SI)
                     {
                      fprintf(stderr,  "\nSolo  se  permite  una  entrada  por
fichero.\n");
                      return ERROR;
                     }
                    else
                     {
                      entrada = SI;
                      laberinto->mapa[x][y] = c;
                      posicion_entrada->x = x;
                      posicion_entrada->y = y;
                     }
                    break;
          case 's':
          case 'S': if (salida == SI)
                     {
                      fprintf(stderr,  "\nSolo  se  permite  una  salida  por
fichero.\n");
                      return ERROR;
                     }
                    else
                     {
                      salida = SI;
                      laberinto->mapa[x][y] = c;
                      coordenada_salida.x = x;
                      coordenada_salida.y = y;
                      if (salida_rodeada(laberinto, coordenada_salida)==SI)
                        salida_imposible=SI;
                     }
                     break;

          case 'x':
          case 'X': laberinto->mapa[x][y] = c;
                    break;

          case 'f':
          case 'F': if (laberinto->ancho == -1)
                     laberinto->ancho = x - 1;
                    else if (laberinto->ancho != x - 1)
                     {
                      fprintf(stderr,"\nLos  anchos  no  concuerdan,  por  favor
revise el archivo.\n");
                      return ERROR;
                     }
                    x = -2;
                    break;

          case  '\n':  fprintf(stderr,  "\nLinea  sin  delimitador  de  final
('F').\n");
                     return ERROR;

          case ' ': laberinto->mapa[x][y] = c;
```

```
                    break;

          default:  fprintf(stderr,  "\nCaracter  invalido,  por  favor  revise  el
fichero\n");
                   return ERROR;
         } /* Fin switch */
       } /* Fin: else */

     if (x==LADO_MAX)
      {
       fprintf(stderr,"\nLaberinto demasiado ancho, por favor reviselo.\n");
       return ERROR;
      }
      x++;
   } /* Fin while */

   if (salida==NO)
    {
     fprintf(stderr, "\nDebe haber alguna salida.");
     return ERROR;
    }

   if (entrada==NO)
    {
     fprintf(stderr, "\nDebe haber alguna entrada.");
     return ERROR;
    }

   laberinto->alto = y-1;
   fclose(ptr);

   if (salida_imposible==SI)
    {
     fprintf(stdout, "\nEl laberinto no tiene solucion.");
     fprintf(stdout, "\nLa salida esta rodeada por X.\n\n");
     return SALIDA_IMPOSIBLE;
    }

 return BIEN;
}

/**************************************************************************
*
*   Funcion: void imprime_laberinto(LABERINTO * laberinto)
*
*   IN: Puntero al laberinto.
*   OUT:
*   MAKE: Imprime el laberinto por pantalla.
*
**************************************************************************/
void imprime_laberinto(LABERINTO * laberinto)
{
 #define AltoPermitido 16

 int x, y;

   for (y=0; y <= laberinto->alto; y++)
    {
     for (x=0; x <= laberinto->ancho; x++)
      {
       if (y) /* Si y es != de 0 */
        {
         /* Paramos cada "AltoPermitido" lineas */
         if ( ((y % AltoPermitido) == 0) && (x==0) )
          {
           fflush(stdin);
           fgetc(stdin); /* Leemos un caracter para detener la pantalla */
           fprintf(stdout, "%c", laberinto->mapa[x][y]);
```

```
          }
         else
           fprintf(stdout, "%c", laberinto->mapa[x][y]);
        }
       else
         fprintf(stdout, "%c", laberinto->mapa[x][y]);
      } /* Fin del segundo For */
     fprintf(stdout, "\n");
    } /* Fin del primer For */
   fprintf(stdout, "\n");
}


/**************************************************************************
*
*   Funcion: void imprime_laberinto_solucion(LABERINTO * laberinto)
*
*   IN: Puntero al laberinto.
*   OUT:
*   MAKE: Imprime la solucion del laberinto por pantalla.
*
**************************************************************************/
void imprime_laberinto_solucion(LABERINTO * laberinto)
{
 int x, y;

 printf("\nPulse una tecla para ver la solucion:");
 fflush(stdin);
 fgetc(stdin);

  for (y = 0; y <= laberinto->alto; y++)
   {
    for (x = 0; x <= laberinto->ancho; x++)
     {
      if (y)
       {
        if ( ((y % AltoPermitido) == 0) && (x==0) )
         {
          fflush(stdin);
          fgetc(stdin);
          if (laberinto->mapa[x][y] != ',')
            fprintf(stdout, "%c", laberinto->mapa[x][y]);
          else
            fprintf(stdout, " ");
         }
        else
         {
          if (laberinto->mapa[x][y] != ',')
            fprintf(stdout, "%c", laberinto->mapa[x][y]);
          else
            fprintf(stdout, " ");
         }
       }
      else
       {
        if (laberinto->mapa[x][y] != ',')
          fprintf(stdout, "%c", laberinto->mapa[x][y]);
        else
          fprintf(stdout, " ");
       }
     }
    fprintf(stdout, "\n");
   }
  fprintf(stdout, "\n");
}
```

```
/*************************************************************************
*
*     Funcion: BOOLEANO es_casilla_permitida(LABERINTO *laberinto, COORDENADA
posicion_actual)
*
*   IN: Puntero al laberinto y la posicion actual.
*    OUT: Si la casilla esta permitida devuelve SI, de lo contrario devuelve
NO.
*   MAKE: Dice si podemos ocupar la casilla del laberinto o no. Podemos ocupar
las
*         casillas que tienen un espacio vacio, son 'E', 'e', 'S' o 's'.
*
*************************************************************************/
BOOLEANO      es_casilla_permitida(LABERINTO      *laberinto,      COORDENADA
posicion_actual)
{
  if  (  (posicion_actual.x  >  laberinto->ancho)  ||  (posicion_actual.y  >
laberinto->alto) )
    return NO;

  if (laberinto->mapa[posicion_actual.x][posicion_actual.y] == 'S' ||
      laberinto->mapa[posicion_actual.x][posicion_actual.y] == ' ')
     return SI;

 return NO;
}

/*************************************************************************
*
*      Funcion:  BOOLEANO  es_casilla_salida(LABERINTO  *laberinto,  COORDENADA
posicion_actual)
*
*   IN: Puntero al laberinto y posicion actual.
*   OUT: Si es la casilla de salida devuelve SI, si no devuelve NO.
*   MAKE: Dice si esa casilla es la salida ('S') o no.
*
*************************************************************************/
BOOLEANO es_casilla_salida(LABERINTO *laberinto, COORDENADA posicion_actual)
{
  if (laberinto->mapa[posicion_actual.x][posicion_actual.y] == 'S')
     return SI;

 return NO;
}


/*************************************************************************
*
*       Funcion:   BOOLEANO   salida_rodeada(LABERINTO   *laberinto,   COORDENADA
coord_salida)
*
*   IN: Puntero al laberinto y la coordenada de salida.
*    OUT: Si la salida es imposible devuelve SI, si no esta rodeada por X
devuelve NO.
*   MAKE: Comprobamos si la casilla de salida esta rodeada por X.
*
*************************************************************************/
BOOLEANO salida_rodeada(LABERINTO *laberinto, COORDENADA coord_salida)
{

 MOVIMIENTO movimiento = ARRIBA;
 COORDENADA posicion_actual;

 while (movimiento != NINGUNO)
  {
   posicion_actual = calcula_nueva_posicion(coord_salida, movimiento);
   if (es_casilla_permitida(laberinto, posicion_actual)==SI)
     return NO;
```

```
   politica_movimiento(&movimiento);
  }

 return SI;
}

/*************************************************************************
*
*   Funcion: MOVIMIENTO politica_movimiento(MOVIMIENTO *ultimo_movimiento)
*
*   IN: Puntero al ultimo movimiento realizado.
*   OUT: Devuelve el movimiento de acuerdo a nuestra politica.
*   MAKE: En base al ultimo movimiento que se ha realizado, nos dice que
*         movimiento debemos intentar ahora: ninguno->arriba, arriba->derecha,
*         derecha->abajo, abajo->izquierda, izquierda->ninguno.
*
*************************************************************************/
MOVIMIENTO politica_movimiento(MOVIMIENTO *ultimo_movimiento)
{
  switch (*ultimo_movimiento)
   {
    case NINGUNO:   return *ultimo_movimiento = ARRIBA;

    case ARRIBA:    return *ultimo_movimiento = DERECHA;

    case DERECHA:   return *ultimo_movimiento = ABAJO;

    case ABAJO:     return *ultimo_movimiento = IZQUIERDA;

    case IZQUIERDA: return *ultimo_movimiento = NINGUNO;
   }

 return *ultimo_movimiento;
}


/*************************************************************************
*
*   Funcion: MOVIMIENTO movimiento_opuesto(MOVIMIENTO movimiento)
*
*   IN: Movimiento del que calcularemos el opuesto.
*   OUT: Devuelve el movimiento opuesto.
*  MAKE: Para un movimiento dado, nos dice su opuesto. Izquierda y derecha son
*     opuestos, arriba y abajo son opuestos. El opuesto de ninguno es ninguno.
*
*************************************************************************/
MOVIMIENTO movimiento_opuesto(MOVIMIENTO movimiento)
{
  switch (movimiento)
   {
    case NINGUNO:   return NINGUNO;

    case ARRIBA:    return ABAJO;

    case DERECHA:   return IZQUIERDA;

    case ABAJO:     return ARRIBA;

    case IZQUIERDA: return DERECHA;
    }

 return NINGUNO;
}


/*************************************************************************
*
*   Funcion: COORDENADA calcula_nueva_posicion(COORDENADA posicion_actual,
```

```
*            MOVIMIENTO siguiente_movimiento)
*
*   IN: La posicion actual y el siguiente movimiento que tenemos que realizar.
*   OUT: La coordenada que se calcula con la posicion actual y el movimiento.
*   MAKE: Esta funcion nos dice que coordenada tendremos si realizamos el
*         movimiento especificado desde la posicion actual.
*
*****************************************************************************/
COORDENADA   calcula_nueva_posicion(COORDENADA   posicion_actual,   MOVIMIENTO
siguiente_movimiento)
{
  switch (siguiente_movimiento)
   {
    case NINGUNO:   return posicion_actual;

    case ARRIBA:    posicion_actual.y--;       /* Fila superior a la actual */
                        return posicion_actual;

    case DERECHA:   posicion_actual.x++;   /* Columna siguiente a la actual */
                        return posicion_actual;

    case ABAJO:     posicion_actual.y++;       /* Fila inferior a la actual */
                        return posicion_actual;

    case IZQUIERDA: posicion_actual.x--;    /* Columna anterior a la actual */
                        return posicion_actual;
    }

 return posicion_actual;
}


/*****************************************************************************
*
*   Funcion: void marca_laberinto(LABERINTO * laberinto, COORDENADA
*            posicion_actual)
*
*   IN: Puntero al laberinto y posicion actual.
*   OUT:
*   MAKE: Esta funcion marca el laberinto, en la posicion en la que estemos,
*         con un punto '.'
*
*****************************************************************************/
void marca_laberinto(LABERINTO * laberinto, COORDENADA posicion_actual)
{
  laberinto->mapa[posicion_actual.x][posicion_actual.y] = '.';
}


/*****************************************************************************
*
*   Funcion: void marca_laberintoMalCamino(LABERINTO * laberinto, COORDENADA
*            posicion_actual)
*
*   IN: Puntero al laberinto y la posicion actual.
*   OUT:
*   MAKE: Funcion que marca el laberinto, en la posicion en la que estemos,
*         con una coma ','
*
*****************************************************************************/
void marca_laberintoMalCamino(LABERINTO * laberinto, COORDENADA
     posicion_actual)
{
  laberinto->mapa[posicion_actual.x][posicion_actual.y] = ',';
}
```

```
/**************************************************************************
*
* Funcion: void marca_con_E(LABERINTO * laberinto, COORDENADA posicion_actual)
*
*   IN: Un puntero al laberinto y la posicion actual
*   OUT:
*   MAKE: Esta funcion marca el laberinto, en la posicion en la que estemos,
*         con una 'E'
*
**************************************************************************/
void marca_con_E(LABERINTO * laberinto, COORDENADA posicion_actual)
{
  laberinto->mapa[posicion_actual.x][posicion_actual.y] = 'E';
}
```

## 8.algorit.c

```
#include <stdio.h>
#include "infopila.h"
#include "lab.h"
#include "pila.h"
#include "tipos.h"
#include "algorit.h"
#include "stdio.h"

/**************************************************************************
*
*   Funcion: ESTADO Algoritmo(LABERINTO *laberinto, COORDENADA coordenada, int
*            parametro, BOOLEANO *solucion)
*
*   IN: Puntero al laberinto, coordenada, un parametro que es el comando y si
*       el laberinto tiene solucion se acumulara en *solucion.
* OUT: Si las cosas marchar correctamente devuelve BIEN, si no devuelve ERROR.
*   MAKE: Funcion para realizar el algoritmo del laberinto.
*
**************************************************************************/
ESTADO Algoritmo(LABERINTO *laberinto, COORDENADA coordenada, int parametro,
       BOOLEANO *solucion)
{
  MOVIMIENTO movimiento = ARRIBA;
  PILA pila;
  char basura[10];
  char ver_mov;             /* El ususario lo elige para ver los movimientos*/


  InicializaPila(&pila);
  if (!laberinto)
    return ERROR;
  Push(&pila, NINGUNO);

  do {
    if (movimiento == NINGUNO)
     {
      marca_laberintoMalCamino(laberinto, coordenada);
      if (PilaVacia(&pila) == NO)
        Pop(&pila, &movimiento);
      coordenada = calcula_nueva_posicion(coordenada,
                   movimiento_opuesto(movimiento));
      }
    else
     {
      if (es_casilla_permitida (laberinto, calcula_nueva_posicion(coordenada,
movimiento)) == SI)
       {
        Push(&pila, movimiento);
        if (parametro==3)
         {
```

```
          imprime_laberinto(laberinto);
          fgets(basura, 10, stdin);
          fflush(stdin);
         }
        coordenada = calcula_nueva_posicion(coordenada, movimiento);
        if (es_casilla_salida(laberinto, coordenada) == SI)
         {
          fprintf(stdout, "\nLaberinto CON solucion\n\n");
          *solucion=SI;
          printf("\nSi desea ver los movimientos escriba 's' o 'S': ");
          ver_mov = fgetc(stdin);
          if ( (ver_mov=='s') || (ver_mov=='S') )
            mostrar_movimientos(&pila);
          printf("\nIntroduzca cualquier caracter para ver la solucion");
          fflush(stdin);
          fgetc(stdin);
          LiberaPila(&pila);
          return BIEN;
         }
        marca_laberinto(laberinto, coordenada);
        movimiento=NINGUNO;
       }
     }

     politica_movimiento(&movimiento);
   } while (PilaVacia(&pila) == NO);

   marca_con_E(laberinto,coordenada);
   fprintf(stdout, "\nLaberinto SIN solucion\n");
   LiberaPila(&pila);

  return BIEN;
}
```

9.main.c

```
#include "algorit.h"
#include "pila.h"
#include "lab.h"
#include "tipos.h"
#include <stdio.h>
#include <string.h>

void AyudaUso(void);

int main(int argc, char *argv[])
{
 BOOLEANO solucion=NO;
 LABERINTO laberinto;
 COORDENADA coordenada;
 int compara;
 char comando[]="-s";

  if ( !((argc==2) || (argc==3)) )
   {
    AyudaUso();
    return ERROR_COMANDO;
   }
  if (argc == 3)
   {
    compara = strcmp(argv[2], comando);
    /* Si no se ha introducido -s como comando mostramos el error */
    if (compara)
     {
      AyudaUso();
      return ERROR_COMANDO;
     }
```

```
   }

  switch (lee_laberinto(argv[1], &laberinto, &coordenada))
   {
    case ERROR:            fprintf(stderr,"\nError al leer el fichero.\n");
                           return ERROR;
    case SALIDA_IMPOSIBLE: imprime_laberinto(&laberinto);
                           return SALIDA_IMPOSIBLE;
   }

  if ( !Algoritmo(&laberinto, coordenada, argc, &solucion) )
   {
    fprintf(stderr,"\nError al pasar el algoritmo, por favor consulte con su
distribuidor.");
    return ERROR;
   }

  fprintf(stdout,"\nEl trayecto realizado ha sido: \n");
  imprime_laberinto(&laberinto);

  if (solucion==SI)
   {
    fprintf(stdout,"\nLa solucion del laberinto es: \n");
    imprime_laberinto_solucion(&laberinto);
   }

 return BIEN;
}


/**************************************************************************
*
*   Funcion: void AyudaUso(void)
*
*   IN:
*   OUT:
*   MAKE: Muestra un mensaje de ayuda para el usuario para que introduzca los
*         comandos correctamente.
*
**************************************************************************
*/
void AyudaUso(void)
{
  fprintf(stderr,"\nEl uso del comando es: <ejecutable> <nombre del laberinto>
[-s]\n");
}
```

Los archivos de prueba con los laberintos para este programa deberían venir adjuntos con este documento. Si hay algún problema pueden visitar mi web y adquirirlos: www.victorsanchez2.net

## PRÁCTICA 4

### OBJETIVOS

El objetivo de esta práctica es implementar la Meta Estructura de Datos (Meta EdD) LISTA y realizar un ejercicio de aplicación de dicho tipo de datos. El ejercicio consistirá en programar la gestión de una serie de *personas* (utilizando la estructura de datos definida en la práctica 2) usando listas para ordenarlas de acuerdo a diversos criterios así como para definir la lista de *amigos* de una persona. La implementación del

tipo de dato será una implementación genérica, es decir no dependiente del tipo de elemento que se va a guardar en la lista.

EJERCICIOS

1. Implementación de la Meta EdD *LISTA* (archivos *lista.h* y *lista.c)*
El archivo *lista.h* contiene las constantes y definiciones de tipos asociadas a la Meta EdD lista y la cabecera de las funciones asociadas a esa Meta EdD.

El archivo lista.c deberá de contener el código que implementa las funciones definidas en lista.h.

2. Escribir un programa de prueba para comprobar el funcionamiento adecuado de la implementación de la Meta EdD LISTA. El programa debe recibir como argumentos por la línea de comandos una serie de cadenas de caracteres e insertarlas en una lista enlazada. Una vez hecho esto debe recorrer la lista para volver a obtener dichas cadenas e imprimirlas por pantalla.

3. Para este ejercicio se utilizará la estructura de datos persona utilizada en la práctica 2, en la cual el campo *amigo* se ha redefinido como *LISTA amigos*. Dicha definición se encuentra en el archivo persona.h que se adjunta.

Implementar también en dicho archivo la función auxiliar LiberarPersona de la práctica 2. El archivo persona.c deberá de contener el código que implementa las funciones definidas en persona.h.

4. Realizar un programa driver para probar las funciones de manipulación de listas. Dicho programa deberá:

A. Leer de un archivo de texto (cuyo nombre se recibirá como el primer argumento pasado al programa) los datos de una serie de personas Realizar un programa que e insertarlos en una lista. Dicha lista se mantendrá en memoria hasta la finalización del programa.
   El formato de este fichero será:
   Nombre\tapellidos\tedad\tDNI\n
   Campos o datos de una persona separados por tabuladores, cada persona termina con salto de carro.
   *Ver fichero personas.txt*

B. Leer de un segundo archivo de texto (cuyo nombre se recibirá como argumento siguiente al anterior) donde estarán contenidos las relaciones de "amistad" entre las personas del fichero del punto A.
   El formato de este fichero será:
   DNIPersona\tDNIAmigo1\tDNIAmigo2...\tDNIAmigoN\n
   Primero está el DNI de una persona y luego los DNI de sus amigos, todo separado por tabuladores, se termina la lista de amigos de una persona con un salto de carro.

Si en la sucesión de DNIs de los amigos hay un DNI no existente (que no estaba en el fichero de personas), ese DNI se ignora y se sigue adelante. Si el DNI de la persona (el primer DNI de cada línea) no existía en el fichero de personas se ignora toda la línea.
*Ver fichero amigos.txt*

C. Una vez cargados los datos de los ficheros anteriores el programa ejecutará un bucle en el cual se podrán elegir las siguientes opciones:
   a. Ordenar por nombre. Se creará una lista ordenada por nombre. Se enseñará por pantalla y a continuación se liberará la memoria de dicha lista.
   b. Ordenar por apellidos. Se creará una lista ordenada por apellidos. Se enseñará por pantalla y a continuación se liberará la memoria de dicha lista.
   c. Ordenar por edad. Se creará una lista ordenada por edad. Se enseñará por pantalla y a continuación se liberará la memoria de dicha lista.
   d. Ordenar por DNI. Se creará una lista ordenada por DNI. Se enseñará por pantalla y a continuación se liberará la memoria de dicha lista.
   e. Mostrar lista de amigos. Se mostrará para cada persona la lista de sus amigos.
   f. Eliminar persona. El programa pedirá identificar a la persona a borrar por medio del DNI, recorrerá la lista para buscar dicha persona y la eliminará de dicha lista. Asimismo se debe comprobar las listas de amigos de cada persona para ver si está presente y eliminarlo de dicha lista. Si hubiese más de una persona con el mismo DNI se eliminarían todas ellas.
   g. Terminar: se libera la memoria utilizada y se termina el programa.

**Ejercicio 2:**

Este ejercico está compuesto de 6 archivos:

1.ejer.h
2.lista.h
3.tipos.h
4.ejer.c
5.lista.c
6.main.c

1.ejer.h

```
#ifndef _EJER_H
#define _EJER_H

ESTADO MeterContenidos(LISTA * plista ,char *  argv[], int argc);
ESTADO MostrarContenidos(LISTA * plista);
void LiberaContenido(void * pcontenido);
ESTADO ImprimeContenido(void *pcontenido);

#endif
```

2.lista.h

```
#ifndef _LISTA_H
#define _LISTA_H
```

```
#include "tipos.h"       /* Donde en tipos.h estan las definiciones de ESTADO
y BOOLEANO */

/* Comienzo de lista.h */

typedef struct NODO {

    void *info;               /*puntero a la información (general) */

    struct NODO *next;        /*puntero al siguiente nodo */

} nodo;

typedef nodo *LISTA;          /*declaración del tipo LISTA */

/* Esta funcion inicializa la lista a una lista vacía */
void InicializaLista(LISTA * lista);

/* Indica si la lista está vacía o no.*/
BOOLEANO ListaVacia(LISTA * lista);

/* Esta funcion obtiene memoria para un nodo de la lista, lo inicializa y
devuelve error si no hay memoria para el nuevo nodo. */
ESTADO ObtenerNodo(nodo ** ppn);

/* Esta funcion inserta en la lista el nodo elemento_actual delante del nodo
elemento_siguiente. Si ya est  dicho nodo en la lista devuelve error. */
ESTADO InsertaElemento(LISTA * lista, nodo * elemento_actual,
                 nodo * elemento_siguiente);

/* Esta funcion elimina de la lista el elemento especificado. Devuelve error
en el
caso de que dicho elemento no se encuentre en la lista. */
ESTADO EliminaElemento(LISTA * lista, nodo * elemento_actual);

/* Esta funcion imprime por pantalla el contenido de la lista en el orden en
el
que están los elementos en la lista. */
void ImprimeLista(LISTA lista);

/* Esta funcion es una modificacion de la anterior que imprime tambien la
lista de amigos.  */
void ImprimeListaEspecial(LISTA lista);

/* Esta funcion va recorriendo la lista, liberando los nodos por los que va
pasando.
Devuelve una lista vacía. */
void LiberaLista(LISTA * lista);


/*Esta funcion busca por toda la lista un nodo y si lo encuntra nos da su
direccion. */
BOOLEANO ExisteNodo(LISTA * plista, nodo * nodoabuscar);

/* Esta funcion se encarga de buscar el elemento anterior a uno dado */
ESTADO BuscaPrevio(LISTA  lista, nodo * elementoAbuscar,
             nodo ** elementoAnterior);

/*Esta funcion se encarga de liberar los nodos */
void LiberaNodo(nodo * pnodo);


/* Mete un nodo con el campo info apuntando a puntero al final de una lista */
ESTADO MeterContenido(LISTA * plista, void *puntero);

/* Fin de lista.h */
#endif
```

## 3.tipos.h

```
#ifndef _TIPOS_H
#define _TIPOS_H


typedef enum {
      ERROR, BIEN
} ESTADO;

typedef enum {
  SI, NO
} BOOLEANO;


/* fin del archivo tipos.h */
#endif
```

## 4.ejer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include "tipos.h"
#include "lista.h"
#include "ejer.h"

/**************************************************************************
*   Funcion: void LiberaContenido(void * pContenido)
*
*   IN: Puntero a void
*   OUT:
*   MAKE: Libera el contenido del puntero que se nos pasa.
*
**************************************************************************/
void LiberaContenido(void * pContenido)
{
  free((void *) pContenido);
  pContenido = NULL;

 return;
}

/**************************************************************************
*   Funcion: ESTADO ImprimeContenido(void *pContenido)
*
*   IN: Puntero a void.
*   OUT: BIEN si se ha podido mostrar el contenido del puntero adecuadamente,
*        ERROR si el puntero pasado es nulo.
*   MAKE: Muestra el contenido de un puntero a void que le pasemos.
*
**************************************************************************/
ESTADO ImprimeContenido(void *pContenido)
{
  if (!pContenido)
    return ERROR;
  fprintf(stdout,"\t%s",(char *) pContenido);

 return BIEN;
}

/**************************************************************************
*   Funcion: ESTADO MostrarContenidos(LISTA *pLista)
*
```

```
*   IN: Un puntero a una lista dada.
*   OUT: BIEN si hemos podido mostrar el contenido de la lista, ERROR si la
*        lista estaba vacia.
*   MAKE: Muestra el contenido de una lista dada.
*
*****************************************************************/
ESTADO MostrarContenidos(LISTA *pLista)
{
     if (!pLista)
      return ERROR;
     ImprimeLista(*pLista);

 return BIEN;
}

/*****************************************************************
*   Funcion: ESTADO MeterContenidos(LISTA * pLista ,char *  argv[], int argc)
*
*   IN: Puntero a una lista, los argumentos y el numero de argumentos.
*   OUT: BIEN si se han introducido los datos correctamente o ERROR si ha
*        habido algun problema.
*   MAKE: Introduce los argumentos dados por el usuario en una lista.
*
*****************************************************************/
ESTADO MeterContenidos(LISTA *pLista ,char *argv[], int argc)
{
 nodo *pNodo, *pFinal;
 int i;

  if (!pLista || !argv )
    return ERROR;
  pFinal=*pLista;
  ObtenerNodo(&pNodo);
  pNodo->info = (char *) argv[1];
  *pLista=pNodo;
  pFinal=pNodo;
  for (i=3; i<= argc; ++i)
   {
    ObtenerNodo(&pNodo);
    pNodo->info = (char *) argv[i-1];
    InsertaElemento(pLista,pNodo,pFinal);
    pFinal=pNodo;
   }

 return BIEN;
}
```

5.lista.c

```
#include "lista.h"
#include "ejer.h"
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <string.h>

/*****************************************************************
*   Funcion: void InicializaLista(LISTA * lista)
*
*   IN: Puntero a una lista.
*   OUT: Nada.
*   MAKE: Inicializa la lista a una lista vacia.
*
*****************************************************************/
void InicializaLista(LISTA *lista)
{
  *lista = NULL;
```

```
 return;
}

/**************************************************************************
*   Funcion: BOOLEANO ListaVacia(LISTA * lista)
*
*   IN: Puntero a una lista.
*   OUT: Devolvemos SI si la lista esta vacia, NO en caso contrario.
*   MAKE: Indica si la lista esta vacia o no.
*
**************************************************************************/
BOOLEANO ListaVacia(LISTA *lista)
{
  if (*lista == NULL)
    return SI;

 return NO;
}

/**************************************************************************
*   Funcion:  ESTADO ObtenerNodo(nodo ** ppn)
*
*   IN: Puntero a puntero a nodo.
*   OUT: BIEN si se consigue reservar memoria correctamente, ERROR si hay
*        problemas
*   MAKE: Reserva memoria para un nodo de la lista y lo inicializa o
*         devuelve error si no hay memoria para el nuevo nodo.
*
**************************************************************************/
ESTADO ObtenerNodo(nodo **ppn)
{
  if (!ppn)
    return ERROR;
  if ((*ppn = (nodo *) malloc(sizeof(nodo))) == NULL)
   {
    fprintf(stderr,"\nError al alocar memoria dinamica, posiblemente memoria
insuficiente\n");
    return ERROR;
   }
  (*ppn)->info = NULL;
  (*ppn)->next = NULL;

 return BIEN;
}

/**************************************************************************
*   Funcion: BOOLEANO ExisteNodo(LISTA *pLista, nodo *NodoABuscar)
*
*   IN: Puntero a una lista donde
*       tenemos que buscar el nodo y un nodo, al cual
*       debemos buscar uno similar y ver si existe o no.
*   OUT: Si ya hay un nodo con los mismos datos devolvemos SI, de lo contrario
*        devolvemos NO.
*   MAKE:  Busca por toda la lista un nodo y si lo encuentra nos da su
*          direccion.
*
**************************************************************************/
BOOLEANO ExisteNodo(LISTA *pLista, nodo *NodoABuscar)
{
 nodo *pNodo;

  if (!pLista || !NodoABuscar)
    return NO;
  if (ListaVacia(pLista) == SI)
    return NO;
  pNodo = *pLista;
  while (ListaVacia(&pNodo) == NO)
```

```
   {
    if (pNodo == NodoABuscar)
      return SI;
    pNodo = pNodo->next;
   }

 return NO;
}

/**********************************************************************
*   Funcion: ESTADO InsertaElemento(LISTA *lista, nodo *elemento_actual, nodo
*            *elemento_siguiente)
*
*   IN: Puntero a la lista donde estan los datos y dos punteros a nodo, el que
*       vamos a insertar y el que estara a continuacion.
*   OUT:
*   MAKE: Inserta en la lista el nodo elemento_actual delante del nodo
*         elemento_siguiente. Si ya esta dicho nodo en la lista devuelve
*         error.
*
**********************************************************************/
ESTADO InsertaElemento(LISTA *lista, nodo *elemento_actual, nodo
*elemento_siguiente)
{
  if (!lista || !elemento_actual || !elemento_siguiente)
    return ERROR;
  if (ListaVacia(lista) == SI)
    return ERROR;
  /* Si ya hay un nodo igual en la lista devolveremos error. */
  if (ExisteNodo(lista, elemento_actual) == SI)
   {
    /* Mostramos un mensaje de error y la persona que lo ha producido. */
    fprintf(stderr,"\nOperacion errronea con la lista, Intento de reutilizar
un nodo.\n");
    return ERROR;
   }
  /* Insertamos el nodo en la lista. */
  elemento_actual->next     = elemento_siguiente->next;
  elemento_siguiente->next = elemento_actual;

 return BIEN;
}

/**********************************************************************
*   Funcion: ESTADO EliminaElemento(LISTA *lista, nodo *elemento_actual)
*
*   IN: Puntero a la lista donde estan los datos y el nodo que queremos
*       eliminar
*   OUT: Devolvemos BIEN si todo ha ido correctamente, ERROR si hemos
*        encontrado algun problema que nos impide eliminar el elemento.
*   MAKE: Elimina de la lista el elemento especificado. Devuelve error en el
*         caso de que dicho elemento no se encuentre en la lista.
*
**********************************************************************/
ESTADO EliminaElemento(LISTA *lista, nodo *elemento_actual)
{
 nodo *pNodo = NULL, *pNodo2 = NULL;

  if (!lista || !elemento_actual)
    return ERROR;
  /* Miramos si el elemento esta al principio */
  if ((*lista) == elemento_actual)
   {
    *lista = elemento_actual->next;
    LiberaNodo(elemento_actual);
    return BIEN;
   }
  /* Buscamos despues */
```

```
  if (!BuscaPrevio(*lista, elemento_actual, &pNodo))
   {
    fprintf(stderr, "\nError en la busqueda, imposible el borrado.\n");
    return ERROR;
   }
  if (pNodo == NULL)
   {
    fprintf(stderr, "\nElemento no encontrado, imposible el borrado.\n");
    return ERROR;
   }
  /* Si hemos encontrado el nodo en la lista lo eliminamos */
  pNodo2 = pNodo->next;
  pNodo->next = (pNodo->next)->next;
  LiberaNodo(pNodo2);

 return BIEN;
}

/**************************************************************************
*   Funcion: ESTADO BuscaPrevio(LISTA lista, nodo *ElementoABuscar, nodo
*                  *ElementoAnterior)
*
*   IN: Una lista con los datos, un puntero a nodo ElementoABuscar del cual
*       devemos buscar su elemento anterior.
*   OUT: Devolvemos ERROR si no se puede encontrar el elemento anterior, BIEN
*        si se ha encontrado satisfactoriamente.
*   MAKE: Funcion que se encarga de buscar el elemento anterior a uno dado.
*
**************************************************************************/
ESTADO BuscaPrevio(LISTA lista, nodo *ElementoABuscar, nodo
**ElementoAnterior)
{

  *ElementoAnterior=NULL;
   if (!lista || !ElementoABuscar)
     return ERROR;
   if (ElementoABuscar == lista)
     return ERROR;
   if (ListaVacia(&lista)==SI)
     return BIEN;
   while (ListaVacia(&lista->next) == NO)
    {
     if ((lista->next) == ElementoABuscar)
      {
       *ElementoAnterior=lista;
       return BIEN;
      }
     lista=lista->next;
    }
    *ElementoAnterior = NULL;

 return BIEN;
}

/**************************************************************************
*   Funcion: void LiberaNodo(nodo *pNodo)
*
*   IN: Puntero a nodo del cual debemos liberar la memoria.
*   OUT: Nada, solo se libera el puntero y se pone a NULL.
*   MAKE: Se encarga de liberar los nodos.
*
**************************************************************************/
void LiberaNodo(nodo *pNodo)
{
  free(pNodo);
  pNodo = NULL;
}
```

```
/**************************************************************************
*   Funcion: void ImprimeLista(LISTA lista)
*
*   IN: Una lista con los datos.
*   OUT: Muestra por pantalla los datos que contiene.
*   MAKE: Imprime por pantalla el contenido de la lista en el orden en el que
*         estan los elementos de la lista.
*
**************************************************************************/
void ImprimeLista(LISTA lista)
{
  while (ListaVacia(&lista))
   {
    ImprimeContenido(lista->info);
    lista = lista->next;
   }
  fprintf(stdout, "\n");

 return;
}

/**************************************************************************
*   Funcion: void LiberaLista(LISTA * lista)
*
*   IN: Puntero a lista.
*   OUT: Nada.
*   MAKE: Esta funcion va recorriendo la lista, liberando los nodos por los
*         que va pasando.
*
**************************************************************************/
void LiberaLista(LISTA *lista)
{
 nodo *pn;

  pn = *lista;
  while (ListaVacia(lista) == NO)
   {
    *lista = pn->next;
    LiberaNodo(pn);
    pn = *lista;
   }

 return;
}

/**************************************************************************
*   Funcion: ESTADO MeterContenido(LISTA *pLista, void *puntero)
*
*   IN: Puntero a una lista y un puntero a void.
*   OUT: BIEN si se mete el contenido correctamente, ERROR si se encontrado
*        algun puntero nulo o no se ha podido reservar memoria.
*   MAKE: Mete un nodo con el campo info apuntando a puntero al final de una
*         lista.
*
**************************************************************************/
ESTADO MeterContenido(LISTA *pLista, void *puntero)
{
 nodo *pn;
 LISTA lista;

  if (!pLista || !puntero)
    return ERROR;
  if (!ObtenerNodo(&pn))
    return ERROR;
  if (ListaVacia(pLista)==SI)
   {
    *pLista=pn;
    pn->info=puntero;
```

```
    return BIEN;
   }
  lista=*pLista;
  while (ListaVacia(&(lista->next)) != SI)
    lista=lista->next;

  lista->next=pn;
  pn->info=puntero;

 return BIEN;
}
```

6.main.c

```
/*****************************************************************************
*
*   Autor: Victor Sanchez2
*   Web: www.victorsanchez2.net
*   Correo: victorss19@eresmas.com
*   Make: Este programa recibe como argumentos por la linea de comandos una
*          serie de cadenas de caracteres y las inserta en una lista enlazada.
*          Despues de esto recorre la lista para imprimirlas por pantalla.
*
*****************************************************************************/

#include "lista.h"
#include "tipos.h"
#include "ejer.h"
#include <stdio.h>

void ayuda(void);  /* Muestra ayuda al usuario para introducir las cadenas */

LISTA lista=NULL;

int main(int argc, char *argv[])
{
  /* Comprobamos el numero de parametros */
  if (argc == 1)
   {
    ayuda();
     return ERROR;
   }

  /* Metemos las cadenas */
  if (!MeterContenidos(&lista, argv, argc))
   {
     fprintf(stderr,"\nError al insertar los contenidos \n");
     return ERROR;
   }
  /* Mostramos los contenidos */
  if (!MostrarContenidos(&lista))
   {
     fprintf(stderr,"\nError al mostrar los contenidos \n");
     return ERROR;
   }

  /* Liberar la lista */
  LiberaLista(&lista);

 return BIEN;
}

/*********************************************************************
*   Funcion: void ayuda()
*
*   IN:
```

```
*   OUT:
*   MAKE: Muestra un mensaje de ayuda para que el usuario introduzca los
*         parametros correctamente.
*
*****************************************************************/
void ayuda()
{
 fprintf(stderr,"\nUso del programa: <ejecutable> <cadena1> <cadena2> ...
\n");
}
```

**Ejercicio 4:**

Este ejercico está compuesto de 6 archivos:

1.lista.h
2.persona.h
3.tipos.h
4.lista.c
5.persona.c
6.main.c

1.lista.h

```
#ifndef _LISTA_H
#define _LISTA_H

#include "tipos.h"    /* En tipos.h estan las definiciones de ESTADO y
BOOLEANO */

/* Comienzo de lista.h */

typedef struct NODO {

    void *info;               /* Puntero a la informacion (general) */

    struct NODO *next;        /* Puntero al siguiente nodo */

} nodo;

typedef nodo *LISTA;          /* Declaracion del tipo LISTA */


/* Inicializa la lista a una lista vacia. */
void InicializaLista(LISTA *lista);

/* Indica si la lista esta vacia o no. */
BOOLEANO ListaVacia(LISTA *lista);

/* Obtiene memoria para un nodo de la lista y lo inicializa o
   devuelve error si no hay memoria para el nuevo nodo. */
ESTADO ObtenerNodo(nodo **ppn);

/* Inserta en la lista el nodo elemento_actual delante del nodo
   elemento_siguiente. Si ya esta dicho nodo en la lista devuelve error. */
ESTADO InsertaElemento(LISTA *lista, nodo *elemento_actual, nodo
*elemento_siguiente);

/* Elimina de la lista el elemento especificado. Devuelve error en el
   caso de que dicho elemento no se encuentre en la lista. */
ESTADO EliminaElemento(LISTA *lista, nodo *elemento_actual);
```

```
/* Imprime por pantalla el contenido de la lista en el orden en el
   que estan los elementos en la lista. */
void ImprimeLista(LISTA lista);

/* Esta funcion es una modificacion de la anterior que imprime tambien la
lista de amigos. */
void ImprimeListaEspecial(LISTA lista);

/* Esta funcion va recorriendo la lista, liberando los nodos por los que va
pasando. */
void LiberaLista(LISTA *lista);

/* Busca por toda la lista un nodo y si lo encuentra nos da su direccion. */
BOOLEANO ExisteNodo(LISTA * pLista, nodo *NodoABuscar);

/* Esta funcion se encarga de buscar el elemento anterior a uno dado. */
ESTADO BuscaPrevio(LISTA  lista, nodo *elementoAbuscar, nodo
**elementoAnterior);

/*Esta funcion se encarga de liberar los nodos. */
void LiberaNodo(nodo *pnodo);

/* Mete un nodo con el campo info apuntando a puntero al final de una lista */
ESTADO MeterContenido(LISTA *plista, void *puntero);

/* Fin de lista.h */
#endif
```

## 2.persona.h

```
#ifndef _PERSONA_H
#define _PERSONA_H

#include "lista.h"

/* Comienzo de persona.h */

/* estructura persona */
typedef struct persona {
    char *nombre;
    char *apellido;
    int edad;
    char DNI[10];
    LISTA amigos;   /* El conjunto de amigos de una persona es una lista */
} PERSONA;

/* lee de un fichero de texto (una linea por persona) y genera la
lista enlazada con los datos leídos, en el orden en el que figuraban
en el fichero. */
LISTA LeerPersonas(char *nombreFichero);

/*  imprime por pantalla los datos de todas las personas (sin incluir
la lista de amigos). */
void ImprimirListaPersonas(LISTA lista);

/* genera una nueva lista, en la cual los nodos están ordenados por nombre.
Los nuevos nodos deben ser alocados dentro de esta función, si bien los
punteros a los que apuntan los campos info deben seguir siendo los mismos. */
LISTA OrdenarPorNombre(LISTA lista);

/* genera una nueva lista, en la cual los nodos están ordenados por apellido.
Los nuevos nodos deben ser alocados dentro de esta función, si bien los
punteros a los que apuntan los campos info deben seguir siendo los mismos. */
LISTA OrdenarPorApellido(LISTA lista);

/* genera una nueva lista, en la cual los nodos están ordenados por DNI.
```

```
Los nuevos nodos deben ser alocados dentro de esta funcion, si bien los
punteros a los que apuntan los campos info deben seguir siendo los mismos. */
LISTA OrdenarPorDNI(LISTA lista);

/* genera una nueva lista, en la cual los nodos están ordenados por nombre.
Los nuevos nodos deben ser alocados dentro de esta función, si bien los
punteros a los que apuntan los campos info deben seguir siendo los mismos. */
LISTA OrdenarPorEdad(LISTA lista);

/* libera los nodos de una lista enlazada.*/
void LiberarListaPersonas(LISTA *lista);

/* Imprime el contenido de una persona */
ESTADO ImprimeContenido(void *contenido);

/* Imprime el contenido de una persona con sus amigos */
ESTADO ImprimeContenidoEspecial(void *contenido);

/* Libera una persona */
ESTADO LiberaPersona(void *contenido);

/*Busca a una persona en una lista */
nodo *BuscaPersona(PERSONA * ppersona, LISTA lista);

/*Resuelve dependecias causadas por la "muerte de una persona" */
ESTADO ResolverDependencias(LISTA *plista, PERSONA * ppersona);

/*Obtenemos memoria para una persona */
ESTADO ObtenerPersona(PERSONA ** pppersona);

/*Busca a una persona por su DNI en una lista */
PERSONA *BuscaDNIPersona(char *DNI, LISTA lista);

/* lee de un fichero de texto con las dependencias de amigos
, en el orden en el que figuraban en el fichero. */
ESTADO LeerAmigos(char *nombreFichero, LISTA lista);

/*  imprime por pantalla los datos de todas las personas (incluyendo
la lista de amigos). */
void ImprimirListaAmigos(LISTA lista);

/* Este proceso hace de frontend entre las funciones y el usuario */
ESTADO Algoritmo(LISTA *lista);

/* Busca posibles errores en los DNI */
ESTADO ErroresLineaDNI(char *linea);

/* Busca posibles errores en los datos de las personas */
ESTADO ErroresLineaPersonas(char *linea);

#endif
```

## 3.tipos.h

```
#ifndef _TIPOS_H
#define _TIPOS_H


typedef enum {
      ERROR, BIEN
} ESTADO;

typedef enum {
  SI, NO
} BOOLEANO;
```

```
/* fin del archivo tipos.h */
#endif
```

## 4.lista.c

```
#include "lista.h"
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <string.h>
#include "persona.h"

/**********************************************************************
*   Funcion: void InicializaLista(LISTA * lista)
*
*   IN: Puntero a una lista.
*   OUT: Nada.
*   MAKE: Inicializa la lista a una lista vacia.
*
**********************************************************************/
void InicializaLista(LISTA *lista)
{
  *lista = NULL;

 return;
}

/**********************************************************************
*   Funcion: BOOLEANO ListaVacia(LISTA * lista)
*
*   IN: Puntero a una lista.
*   OUT: Devolvemos SI si la lista esta vacia, NO en caso contrario.
*   MAKE: Indica si la lista esta vacia o no.
*
**********************************************************************/
BOOLEANO ListaVacia(LISTA *lista)
{
  if (*lista == NULL)
    return SI;

 return NO;
}


/**********************************************************************
*   Funcion:  ESTADO ObtenerNodo(nodo ** ppn)
*
*   IN: Puntero a puntero a nodo.
*   OUT: BIEN si se consigue reservar memoria correctamente, ERROR si hay
*        problemas
*   MAKE: Reserva memoria para un nodo de la lista y lo inicializa o
*         devuelve error si no hay memoria para el nuevo nodo.
*
**********************************************************************/
ESTADO ObtenerNodo(nodo **ppn)
{
  if (!ppn)
    return ERROR;
  if ((*ppn = (nodo *) malloc(sizeof(nodo))) == NULL)
   {
    fprintf(stderr,"\nError al alocar memoria dinamica, posiblemente memoria
insuficiente\n");
    return ERROR;
   }
  (*ppn)->info = NULL;
  (*ppn)->next = NULL;
```

```
 return BIEN;
}

/**********************************************************************
*   Funcion: BOOLEANO ExisteNodo(LISTA *pLista, nodo *NodoABuscar)
*
*   IN: Puntero a una lista donde tenemos que buscar el nodo y un nodo, al
cual
*       debemos buscar uno similar y ver si existe o no.
*   OUT: Si ya hay un nodo con los mismos datos devolvemos SI, de lo contrario
*        devolvemos NO.
*   MAKE:  Busca por toda la lista un nodo y si lo encuentra nos da su
direccion.
*
**********************************************************************/
BOOLEANO ExisteNodo(LISTA *pLista, nodo *NodoABuscar)
{
 nodo *pNodo;

  if (!pLista || !NodoABuscar)
    return NO;
  if (ListaVacia(pLista) == SI)
    return NO;
  pNodo = *pLista;
  while (ListaVacia(&pNodo) == NO)
   {
    /* Si los todos los datos son iguales es que ya existe esa persona */
    if (
      ( (*(((PERSONA *)pNodo->info)->nombre))   == (*(((PERSONA *)
NodoABuscar->info)->nombre)))   &&
      ( (*(((PERSONA *)pNodo->info)->apellido)) == (*(((PERSONA *)
NodoABuscar->info)->apellido))) &&
      ( ( (((PERSONA *)pNodo->info)->edad))     == ( (((PERSONA *)
NodoABuscar->info)->edad)))     &&
      ( (*(((PERSONA *)pNodo->info)->DNI))      == (*(((PERSONA *)
NodoABuscar->info)->DNI)))
       )
      return SI;

    pNodo = pNodo->next;
   }

 return NO;
}

/**********************************************************************
*   Funcion: ESTADO InsertaElemento(LISTA *lista, nodo *elemento_actual, nodo
*elemento_siguiente)
*
*   IN: Puntero a la lista donde estan los datos y dos punteros a nodo, el que
*       vamos a insertar y el que estara a continuacion.
*   OUT:
*   MAKE: Inserta en la lista el nodo elemento_actual delante del nodo
*         elemento_siguiente. Si ya esta dicho nodo en la lista devuelve
error.
*
**********************************************************************/
ESTADO InsertaElemento(LISTA *lista, nodo *elemento_actual, nodo
*elemento_siguiente)
{
  if (!lista || !elemento_actual || !elemento_siguiente)
    return ERROR;
  if (ListaVacia(lista) == SI)
    return ERROR;
  /* Si ya hay un nodo igual en la lista devolveremos error. */
  if (ExisteNodo(lista, elemento_actual) == SI)
   {
```

```
    /* Mostramos un mensaje de error y la persona que lo ha producido. */
    fprintf(stderr,"\nOperacion errronea con la lista, Intento de reutilizar
un nodo.\n");
    fprintf(stderr,"\n%s %s es un usuario ya existente. Omitimos la
repeticion.\n",
                ((PERSONA*)elemento_actual->info)->nombre,
                ((PERSONA*)elemento_actual->info)->apellido);
    return ERROR;
   }
  /* Insertamos el nodo en la lista. */
  elemento_actual->next     = elemento_siguiente->next;
  elemento_siguiente->next = elemento_actual;

 return BIEN;
}

/*****************************************************************
*   Funcion: ESTADO EliminaElemento(LISTA *lista, nodo *elemento_actual)
*
*   IN: Puntero a la lista donde estan los datos y el nodo que queremos
*       eliminar
*   OUT: Devolvemos BIEN si todo ha ido correctamente, ERROR si hemos
*        encontrado algun problema que nos impide eliminar el elemento.
*   MAKE: Elimina de la lista el elemento especificado. Devuelve error en el
*         caso de que dicho elemento no se encuentre en la lista.
*
*****************************************************************/
ESTADO EliminaElemento(LISTA *lista, nodo *elemento_actual)
{
 nodo *pNodo = NULL, *pNodo2 = NULL;

  if (!lista || !elemento_actual)
    return ERROR;
  /* Miramos si el elemento esta al principio */
  if ((*lista) == elemento_actual)
   {
    *lista = elemento_actual->next;
    LiberaNodo(elemento_actual);
    return BIEN;
   }
  /* Buscamos despues */
  if (!BuscaPrevio(*lista, elemento_actual, &pNodo))
   {
    fprintf(stderr, "\nError en la busqueda, imposible el borrado.\n");
    return ERROR;
   }
  if (pNodo == NULL)
   {
    fprintf(stderr, "\nElemento no encontrado, imposible el borrado.\n");
    return ERROR;
   }
  /* Si hemos encontrado el nodo en la lista lo eliminamos */
  pNodo2 = pNodo->next;
  pNodo->next = (pNodo->next)->next;
  LiberaNodo(pNodo2);

 return BIEN;
}

/*****************************************************************
*   Funcion: ESTADO BuscaPrevio(LISTA lista, nodo *ElementoABuscar, nodo
**ElementoAnterior)
*
*   IN: Una lista con los datos, un puntero a nodo ElementoABuscar del cual
*       devemos buscar su elemento anterior.
*   OUT: Devolvemos ERROR si no se puede encontrar el elemento anterior, BIEN
*        si se ha encontrado satisfactoriamente.
*   MAKE: Funcion que se encarga de buscar el elemento anterior a uno dado.
```

```
*
*****************************************************************/
ESTADO BuscaPrevio(LISTA lista, nodo *ElementoABuscar, nodo
**ElementoAnterior)
{

  *ElementoAnterior=NULL;
   if (!lista || !ElementoABuscar)
     return ERROR;
   if (ElementoABuscar == lista)
     return ERROR;
   if (ListaVacia(&lista)==SI)
     return BIEN;
   while (ListaVacia(&lista->next) == NO)
    {
     if ((lista->next) == ElementoABuscar)
      {
       *ElementoAnterior=lista;
       return BIEN;
      }
     lista=lista->next;
    }
    *ElementoAnterior = NULL;

 return BIEN;
}

/*****************************************************************
*   Funcion: void LiberaNodo(nodo *pNodo)
*
*   IN: Puntero a nodo del cual debemos liberar la memoria.
*   OUT: Nada, solo se libera el puntero y se pone a NULL.
*   MAKE: Se encarga de liberar los nodos.
*
*****************************************************************/
void LiberaNodo(nodo *pNodo)
{
  free(pNodo);
  pNodo = NULL;
}

/*****************************************************************
*   Funcion: void ImprimeLista(LISTA lista)
*
*   IN: Una lista con los datos.
*   OUT: Muestra por pantalla los datos que contiene.
*   MAKE: Imprime por pantalla el contenido de la lista en el orden en el que
*         estan los elementos de la lista.
*
*****************************************************************/
void ImprimeLista(LISTA lista)
{
  while (ListaVacia(&lista))
   {
    ImprimeContenido(lista->info);
    lista = lista->next;
   }
  fprintf(stdout, "\n");

 return;
}

/*****************************************************************
*   Funcion: void ImprimeListaEspecial(LISTA lista)
*
*   IN: Una lista con los datos.
*   OUT: Nada. Muestra las personas con sus amigos.
*   MAKE: Esta funcion es una modificacion de la anterior que imprime tambien
```

```
*         la lista de amigos.
*
*****************************************************************/
void ImprimeListaEspecial(LISTA lista)
{
 int i=1;

  while (ListaVacia(&lista))
   {
    ImprimeContenidoEspecial(lista->info);
    lista = lista->next;
    /* Vamos parando la pantalla cada 4 personas mostradas. */
    if ( (i++ % 4) == 0)
     {
      fprintf(stdout,"\nPulse enter para continuar.\n");
      fgetc(stdin);
     }
   }
  fprintf(stdout, "\n");

 return;
}

/*****************************************************************
*   Funcion: void LiberaLista(LISTA * lista)
*
*   IN: Puntero a lista.
*   OUT: Nada.
*   MAKE: Esta funcion va recorriendo la lista, liberando los nodos por los
que
*         va pasando.
*
*****************************************************************/
void LiberaLista(LISTA *lista)
{
 nodo *pn;

  pn = *lista;
  while (ListaVacia(lista) == NO)
   {
    *lista = pn->next;
    LiberaNodo(pn);
    pn = *lista;
   }

 return;
}

/*****************************************************************
*   Funcion: ESTADO MeterContenido(LISTA *pLista, void *puntero)
*
*   IN: Puntero a una lista y un puntero a void.
*   OUT:
*   MAKE: Mete un nodo con el campo info apuntando a puntero al final de una
*         lista.
*
*****************************************************************/
ESTADO MeterContenido(LISTA *pLista, void *puntero)
{
 nodo *pn;
 LISTA lista;

  if (!pLista || !puntero)
    return ERROR;
  if (!ObtenerNodo(&pn))
    return ERROR;
  if (ListaVacia(pLista)==SI)
   {
```

```
    *pLista=pn;
    pn->info=puntero;
    return BIEN;
   }
  lista=*pLista;
  while (ListaVacia(&(lista->next)) != SI)
    lista=lista->next;

  lista->next=pn;
  pn->info=puntero;

 return BIEN;
}
```

5.persona.c

```
#include "lista.h"
#include "persona.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <string.h>

#define LONG_NOM 50
#define MAX_LENGTH 1024
#define MAX_CARACTERES 30
#define MAX_LONG 50
#define LONGITUD 100

/*********************************************************************
*   Funcion: void ImprimirListaPersonas(LISTA lista)
*
*   IN: Una lista.
*   OUT: Llama a la funcion Imprimelista(lista) que mostrara la lista.
*   MAKE: Imprime por pantalla los datos de todas las personas (sin incluir
*         la lista de amigos).
*
*********************************************************************/
void ImprimirListaPersonas(LISTA lista)
{
  ImprimeLista(lista);
}

/*********************************************************************
*   Funcion: void ImprimirListaAmigos(LISTA lista)
*
*   IN: Una Lista.
*   OUT: Llama a la funcion ImprimeListaEspecial que la mostrara con los
*        amigos.
*   MAKE: Imprime por pantalla los datos de todas las personas (incluyendo
*         la lista de amigos.
*
*********************************************************************/
void ImprimirListaAmigos(LISTA lista)
{
  ImprimeListaEspecial(lista);
}

/*********************************************************************
*   Funcion: void LiberarListaPersonas(LISTA *lista)
*
*   IN:
*   OUT:
*   MAKE: Libera los nodos de una lista enlazada.
*
```

```
*********************************************************************/
void LiberarListaPersonas(LISTA *lista)
{
  while (ListaVacia(lista) != SI)
   {
    LiberaPersona((*lista)->info);
    (*lista) = (*lista)->next;
   }
  LiberaLista(lista);
}

/*********************************************************************
*   Funcion: ESTADO ImprimeContenido(void *contenido)
*
*   IN: Puntero a void.
*   OUT:
*   MAKE: Imprime el contenido de una persona.
*
*********************************************************************/
ESTADO ImprimeContenido(void *contenido)
{
 PERSONA *pPersona;

  pPersona = (PERSONA *) contenido;
  fprintf(stdout, "%s    \t%s    \t%d\t%s\n", pPersona->nombre, pPersona-
>apellido, pPersona->edad, pPersona->DNI);

 return BIEN;
}

/*********************************************************************
*   Funcion: ESTADO ImprimeContenidoEspecial(void *contenido)
*
*   IN:
*   OUT:
*   MAKE: Imprime el contenido de una persona con sus amigos.
*
*********************************************************************/
ESTADO ImprimeContenidoEspecial(void *contenido)
{
 PERSONA *pPersona;

  pPersona = (PERSONA *) contenido;
  fprintf(stdout, "%s    \t%s    \t%d\t%s\n", pPersona->nombre, pPersona-
>apellido, pPersona->edad, pPersona->DNI);
  fprintf(stdout, "Sus Amigos son: \n");
  ImprimeLista(pPersona->amigos);
  fprintf(stdout, "\n\n");

 return BIEN;
}

/*********************************************************************
*   Funcion:  ESTADO LiberaPersona(void *contenido)
*
*   IN:
*   OUT:
*   MAKE: Libera una persona.
*
*********************************************************************/
ESTADO LiberaPersona(void *contenido)
{
 PERSONA *pPersona;

  pPersona = (PERSONA *) contenido;
  free(pPersona->nombre);
  free(pPersona->apellido);
  LiberaLista(&(pPersona->amigos));
```

```
 return BIEN;
}

/**********************************************************************
*   Funcion: nodo *BuscaPersona(PERSONA *pPersona, LISTA lista)
*
*   IN: Puntero a una estructura de tipo PERSONA y una lista.
*   OUT: Devolvemos la lista si se ha encontrado a la persona, NULL si
*        no hemos encontrado a nadie.
*   MAKE: Busca a una persona en una lista.
*
**********************************************************************/
nodo *BuscaPersona(PERSONA *pPersona, LISTA lista)
{
  while (ListaVacia(&lista) != SI)
   {
    if (lista->info == (PERSONA *) pPersona)
      return lista;
    lista = lista->next;
   }

 return NULL; /* Si no se ha encontrado a la persona */
}

/**********************************************************************
*   Funcion: ESTADO ResolverDependencias(LISTA *pLista, PERSONA *pPersona)
*
*   IN: Puntero a una lista y un puntero a una estructura de tipo PERSONA.
*   OUT: BIEN si todo ha ido correctamente, ERROR si algun puntero es NULL.
*   MAKE: Resuelve dependecias causadas por la "muerte de una persona".
*
**********************************************************************/
ESTADO ResolverDependencias(LISTA *pLista, PERSONA *pPersona)
{
 nodo *pNodo = NULL;
 LISTA lista;

  if (!pLista || !pPersona)
    return ERROR;
  lista = *pLista;
  while (ListaVacia(&lista) != SI)
   {
    /* La persona que eliminamos va a dejar de aparecer en la lista
       de amigos de las personas que le tuviesen como amigo */
    pNodo = BuscaPersona(pPersona, ((PERSONA *) (lista->info))->amigos);
    if (pNodo != NULL)
      EliminaElemento(&(((PERSONA *) (lista->info))->amigos), pNodo);
    lista = lista->next;
   }

 return BIEN;
}

/**********************************************************************
*   Funcion: LISTA OrdenarPorNombre(LISTA lista)
*
*   IN: Una lista con los datos de todas las personas.
*   OUT: Una nueva lista con las personas ordenadas por nombre.
*   MAKE: Genera una nueva lista, en la cual los nodos estan ordenador por
*         nombre. Los nuevos nodos deben ser alocados dentro de esta funcion,
*         si bien los punteros a los que apuntan los campos info deben seguir
*         siendo los mismos.
*
**********************************************************************/
LISTA OrdenarPorNombre(LISTA lista)
{
 BOOLEANO continuar;
```

```
 nodo *pNodo, *pnord;
 LISTA listaordenada;

  if (ListaVacia(&lista) == SI)
     return NULL;
  InicializaLista(&listaordenada);
  /* Metemos el primer elemento */
  ObtenerNodo(&pNodo);
  pNodo->info = lista->info;
  listaordenada = pNodo;
  lista = lista->next;

  /* Continuamos con los demas */
  while (ListaVacia(&lista) != SI)
   {
    pnord = listaordenada;
    continuar = SI;
    while (continuar == SI)
     {
      /* Comprobamos si estamos al principio y es menor */
      if ((pnord == listaordenada) &&
          (strcmp( ((PERSONA *) (lista->info))->nombre,
                   ((PERSONA *) (pnord->info))->nombre) < 0))
       {
        ObtenerNodo(&pNodo);
        pNodo->info = lista->info;
        pNodo->next = listaordenada;
        listaordenada = pNodo;
        continuar = NO;
       }
      /* Comprobamos que no estemos al final */
      else if (pnord->next == NULL)
       {
        ObtenerNodo(&pNodo);
        pNodo->info = lista->info;
        InsertaElemento(&listaordenada, pNodo, pnord);
        continuar = NO;
       }
      /* En los demas casos */
      else if (strcmp( ((PERSONA *) (lista->info))->nombre,
                       ((PERSONA *) ((pnord->next)->info))-> nombre) < 0)
       {
        ObtenerNodo(&pNodo);
        pNodo->info = lista->info;
        InsertaElemento(&listaordenada, pNodo, pnord);
        continuar = NO;
       }
      pnord = pnord->next;
     }
     lista = lista->next;
   }

 return listaordenada;
}


/*********************************************************************
*   Funcion: LISTA OrdenarPorApellido(LISTA lista)
*
*   IN: Una lista con los datos de todas las persona.
*   OUT: Una nueva lista con las personas ordenadas por apellido.
*   MAKE: Genera una nueva lista, en la cual los nodos estan ordenados por
*         apellido. Los nuevos nodos deben ser alocados dentro de esta
funcion,
*         si bien los punteros a los que apuntan los campos info deben seguir
*         siendo los mismos.
*
*********************************************************************/
```

```
LISTA OrdenarPorApellido(LISTA lista)
{
 BOOLEANO continuar;
 nodo *pNodo, *pnord;
 LISTA listaordenada;

  if (ListaVacia(&lista) == SI)
    return NULL;
  InicializaLista(&listaordenada);
  /* Metemos el primer elemento */
  ObtenerNodo(&pNodo);
  pNodo->info = lista->info;
  listaordenada = pNodo;
  lista = lista->next;

  /* Seguimos con los demas */
  while (ListaVacia(&lista) != SI)
   {
    pnord = listaordenada;
    continuar = SI;
    while (continuar == SI)
     {
      /* Comprobamos si estamos al principio y es menor */
      if ((pnord == listaordenada) &&
         (strcmp (((PERSONA *) (lista->info))->apellido,
                  ((PERSONA *) (pnord->info))->apellido) < 0))
       {
        ObtenerNodo(&pNodo);
        pNodo->info = lista->info;
        pNodo->next = listaordenada;
        listaordenada = pNodo;
        continuar = NO;
       }
      /* Comprobamos que no estemos al final */
      else if (pnord->next == NULL)
       {
        ObtenerNodo(&pNodo);
        pNodo->info = lista->info;
        InsertaElemento(&listaordenada, pNodo, pnord);
        continuar = NO;
       }
      /* En los demas casos */
      else if (strcmp(((PERSONA *) (lista->info))->apellido,
                      ((PERSONA *) ((pnord->next)->info))->apellido) < 0)
       {
        ObtenerNodo(&pNodo);
        pNodo->info = lista->info;
        InsertaElemento(&listaordenada, pNodo, pnord);
        continuar = NO;
       }
      pnord = pnord->next;
     } /* Fin: while (continuar == SI) */
    lista = lista->next;
   } /* Fin: while (ListaVacia(&lista) != SI) */

 return listaordenada;
}

/*********************************************************************
*   Funcion: LISTA OrdenarPorDNI(LISTA lista)
*
*   IN: Una lista con los datos de todas las persona.
*   OUT: Una nueva lista con las personas ordenadas por DNI.
*   MAKE: Genera una nueva lista, en la cual los nodos estan ordenados por
*         DNI. Los nuevos nodos deben ser alocados dentro de esta funcion, si
*         bien los punteros a los que apuntan los campos info deben seguir
*         siendo los mismos.
*
```

```
*********************************************************************/
LISTA OrdenarPorDNI(LISTA lista)
{
 BOOLEANO continuar;
 nodo *pNodo, *pnord;
 LISTA listaordenada;
 float f1, f2, f3; /* Para comparar los DNI de diferente longitud */

  if (ListaVacia(&lista) == SI)
    return NULL;
  InicializaLista(&listaordenada);
  /* Metemos el primer elemento */
  ObtenerNodo(&pNodo);
  pNodo->info = lista->info;
  listaordenada = pNodo;
  lista = lista->next;

  /* Seguimos con los demas */
  while (ListaVacia(&lista) != SI)
   {
    pnord = listaordenada;
    continuar = SI;
    if (strlen(((PERSONA *) (lista->info))->DNI) != strlen(((PERSONA *)
(pnord->info))->DNI))
     {
      while (continuar == SI)
       {
     f1 = atof(((PERSONA *) (lista->info))->DNI);
     f2 = atof(((PERSONA *) (pnord->info))->DNI);
     f3 = atof(((PERSONA *) ((pnord->next)->info))->DNI);
     /* Si estamos al principio... */
     if ((pnord == listaordenada) && (f1<f2))
      {
       ObtenerNodo(&pNodo);
       pNodo->info = lista->info;
       pNodo->next = listaordenada;
       listaordenada = pNodo;
       continuar = NO;
      }
     else if (pnord->next == NULL)
       {
        ObtenerNodo(&pNodo);
        pNodo->info = lista->info;
        InsertaElemento(&listaordenada, pNodo, pnord);
        continuar = NO;
       }
     else if (f1 < f3)
       {
        ObtenerNodo(&pNodo);
        pNodo->info = lista->info;
        InsertaElemento(&listaordenada, pNodo, pnord);
        continuar = NO;
       }
      pnord = pnord->next;
       }

     }
    while (continuar == SI)
     {
      /* Comprobamos si estamos al principio y es menor */
      if ((pnord == listaordenada) &&
          (strcmp (((PERSONA *) (lista->info))->DNI,
                   ((PERSONA *) (pnord->info))->DNI) < 0))
       {
        ObtenerNodo(&pNodo);
        pNodo->info = lista->info;
        pNodo->next = listaordenada;
        listaordenada = pNodo;
```

```
        continuar = NO;
       }
      /* Comprobamos que no estemos al final */
      else if (pnord->next == NULL)
       {
        ObtenerNodo(&pNodo);
        pNodo->info = lista->info;
        InsertaElemento(&listaordenada, pNodo, pnord);
        continuar = NO;
       }
      /* En los demas casos */
      else if (strcmp(((PERSONA *) (lista->info))->DNI,
                      ((PERSONA *) ((pnord->next)->info))->DNI) < 0)
       {
        ObtenerNodo(&pNodo);
        pNodo->info = lista->info;
        InsertaElemento(&listaordenada, pNodo, pnord);
        continuar = NO;
       }
      pnord = pnord->next;
     } /* Fin: while (continuar == SI) */
    lista = lista->next;
   } /* Fin: while (ListaVacia(&lista) != SI) */
  return listaordenada;
}

/*********************************************************************
*   Funcion: LISTA OrdenarPorEdad(LISTA lista)
*
*   IN: Una lista con los datos de todas las persona.
*   OUT: Una nueva lista con las personas ordenadas por edad.
*   MAKE: Genera una nueva lista, en la cual los nodos estan ordenados por
edad.
*         Los nuevos nodos deben ser alocados dentro de esta funcion, si bien
*         los punteros a los que apuntan los campos info deben seguir siendo
*         los mismos.
*
*********************************************************************/
LISTA OrdenarPorEdad(LISTA lista)
{
 BOOLEANO continuar;
 nodo *pNodo, *pnord;
 LISTA listaordenada;

  if (ListaVacia(&lista) == SI)
    return NULL;
  InicializaLista(&listaordenada);
  /* Metemos el primer elemento */
  ObtenerNodo(&pNodo);
  pNodo->info = lista->info;
  listaordenada = pNodo;
  lista = lista->next;

  /* Seguimos con los demas */
  while (ListaVacia(&lista) != SI)
   {
    pnord = listaordenada;
    continuar = SI;
    while (continuar == SI)
     {
      /* Comprobamos si estamos al principio y es menor */
       if ((pnord == listaordenada) && (((PERSONA *) (lista->info))->edad <
                                        ((PERSONA *) (pnord->info))-
>edad))
        {
         ObtenerNodo(&pNodo);
         pNodo->info = lista->info;
         pNodo->next = listaordenada;
```

```
          listaordenada = pNodo;
          continuar = NO;
         }
        /* Comprobamos que no estemos al final */
        else if (pnord->next == NULL)
         {
          ObtenerNodo(&pNodo);
          pNodo->info = lista->info;
          InsertaElemento(&listaordenada, pNodo, pnord);
          continuar = NO;
         }
        /* En los demas casos */
        else if (((PERSONA *) (lista->info))->edad <
                 ((PERSONA *) ((pnord->next)->info))->edad)
         {
          ObtenerNodo(&pNodo);
          pNodo->info = lista->info;
          InsertaElemento(&listaordenada, pNodo, pnord);
          continuar = NO;
         }
        pnord = pnord->next;
      }
     lista = lista->next;
    }

 return listaordenada;
}

/*********************************************************************
*   Funcion: LISTA LeerPersonas(char *nombreFichero)
*
*   IN: Nombre del fichero de donde debemos leer los datos.
*   OUT: Una lista con los datos, o ERROR si no lo hemos conseguido.
*   MAKE: Lee de un fichero de texto (una linea por persona) y genera la lista
*         enlazada con los datos leidos, en el orden en el que figuraban en el
*         fichero.
*
*********************************************************************/
LISTA LeerPersonas(char *nombreFichero)
{
 FILE *ptr;
 char linea[MAX_LENGTH];
 PERSONA *pPersona;
 nodo *pNodo, *pfinal;
 LISTA lista;

  if (!nombreFichero)
     return ERROR;

  /* Abrimos el archivo para lectura */
  ptr = fopen(nombreFichero, "r");
  if (!ptr)
   {
    fprintf(stderr,"\nError al leer el archivo dado, por favor revise rutas y
permisos.");
    return ERROR;
   }
  /* Sacamos la informacion */
  if (fgets(linea, MAX_LENGTH, ptr) == NULL)
    return NULL;
  /* Buscamos los posibles erroes que se puedan producir en los datos */
  ErroresLineaPersonas(linea);
  if (!ObtenerPersona(&pPersona))
    return NULL;
  sscanf(linea, "%s\t%s\%d\t%s\n",pPersona->nombre, pPersona->apellido,
&pPersona->edad, pPersona->DNI);
  if (ObtenerNodo(&pNodo) == ERROR)
    return NULL;
```

```
  pNodo->info = pPersona;
  lista = pNodo;
  pfinal = lista;
  /* Sacamos los demas */
  while (fgets(linea, MAX_LENGTH, ptr) != NULL)
   {
    ErroresLineaPersonas(linea);
    if (!ObtenerPersona(&pPersona))
      return NULL;
    sscanf(linea, "%s\t%s\%d\t%s\n", pPersona->nombre, pPersona->apellido,
&pPersona->edad, pPersona->DNI);
    if (ObtenerNodo(&pNodo) == ERROR)
      return NULL;
    pNodo->info = pPersona;
    InsertaElemento(&lista, pNodo, pfinal);
    pfinal = pNodo;
   }
  fclose(ptr);

 return lista;
}

/**********************************************************************
*   Funcion: ESTADO ObtenerPersona(PERSONA **ppPersona)
*
*   IN: Puntero a puntero a una estructura de tipo PERSONA.
*   OUT: BIEN si se ha conseguido reservar memoria, ERROR si no podemos.
*   MAKE: Obtenemos memoria para una persona.
*
**********************************************************************/
ESTADO ObtenerPersona(PERSONA **ppPersona)
{
  if (!ppPersona)
    return ERROR;
  if ((*ppPersona = (PERSONA *) malloc(sizeof(PERSONA))) == NULL)
   {
    fprintf(stderr,"\nError al alocar memoria dinamica, posiblemente memoria
insuficiente.\n");
    return ERROR;
   }
  if (((*ppPersona)->nombre = (char *) malloc(LONG_NOM * sizeof(char))) ==
NULL)
   {
    fprintf(stderr,"\nError al alocar memoria dinamica, posiblemente memoria
insuficiente.\n");
    return ERROR;
   }
  if (((*ppPersona)->apellido = (char *) malloc(LONG_NOM * sizeof(char))) ==
NULL)
   {
    fprintf(stderr,"\nError al alocar memoria dinamica, posiblemente memoria
insuficiente.\n");
    return ERROR;
   }
  InicializaLista(&(*ppPersona)->amigos);

 return BIEN;
}

/**********************************************************************
*   Funcion: PERSONA *BuscaDNIPersona(char *DNI, LISTA lista)
*
*   IN: DNI de la persona y la lista donde debemos buscar.
*   OUT: Devolvemos la persona si la encontramos, NULL si no la hemos hallado.
*   MAKE: Busca a una persona por su DNI en una lista.
*
**********************************************************************/
PERSONA *BuscaDNIPersona(char *DNI, LISTA lista)
```

```
{
  if (!DNI)
    return NULL;
  if (ListaVacia(&lista) == SI)
    return NULL;
  while (ListaVacia(&lista) != SI)
   {
    /* Vamos comparando el DNI que nos dan con los de la lista */
    if (strcmp(DNI, ((PERSONA *) (lista->info))->DNI) == 0)
      return (PERSONA *) lista->info;
    lista = lista->next;
   }
  fprintf(stderr,"\nEl DNI %s no se encuentra en el archivo de
personas.",DNI);
  fprintf(stderr,"\nRevise el fichero si quiere ver todo correctamente.\n");

 return NULL;
}

/*********************************************************************
*   Funcion: ESTADO LeerAmigos(char *nombreFichero, LISTA lista)
*
*   IN: Nombre del fichero donde deben estar los amigos y una lista.
*   OUT:
*   MAKE: Lee de un fichero de texto con las dependencias de amigos en el
orden
*         en el que figuraban en el fichero.
*
*********************************************************************/
ESTADO LeerAmigos(char *nombreFichero, LISTA lista)
{
 BOOLEANO continuar, finFichero = NO;
 FILE *ptr;
 char *linea, DNI[10], *pChar,  *pLiberador;
 PERSONA *pPersona = NULL, *pAmigo = NULL;

  linea = (char *) malloc(MAX_LENGTH * sizeof(char));
  pLiberador=linea;
  /* Abrimos el archivo para lectura. */
  ptr = fopen(nombreFichero, "r");
  if (!ptr)
   {
    fprintf(stderr,"\nError al leer el archivo dado, por favor revise rutas y
permisos.");
    free(pLiberador);
    return ERROR;
   }
  /* Sacamos la informacion del archivo. */
  if (fgets(linea, MAX_LENGTH, ptr) == NULL)
    finFichero = SI;
  /* Vemos si hay errores en el DNI y mostramos al usuario el correspondiente
     mensaje de alarma (sin terminar el programa) */
  ErroresLineaDNI(linea);
  while (finFichero != SI)
   {
    continuar = NO;
    /* Sacamos el primer DNI. */
    if ((pChar = strchr(linea, '\t')) != NULL)
     {
      sscanf(linea, "%s\t", DNI);
      /* Obtenemos los amigos */
      linea = pChar + 1;
      continuar = SI;
     }
    /* Buscamos la persona a la que se refiere. */
    if (continuar == SI)
      if ((pPersona = BuscaDNIPersona(DNI, lista)) == NULL)
        fprintf(stderr,"\nSe omitira la lista de amigos del DNI erroneo.\n");
```

```
      else
       {
          /* Actuamos con ella.*/
          while ((pChar = strchr(linea, '\t')) != NULL)
           {
            sscanf(linea, "%s\t", DNI);
            /* No pueden tener el mismo DNI la persona y el amigo */
            if (*(pPersona->DNI) == *DNI)
          {
           fprintf(stderr,"\nUna persona no puede ser su propio amigo.");
           fprintf(stderr,"\nEl DNI %s es el mismo.", DNI);
           /*return ERROR;*/
          }
            linea = pChar + 1;
            if ((pAmigo = BuscaDNIPersona(DNI, lista)) != NULL)
           if (!MeterContenido(&(pPersona->amigos), (void *) pAmigo))
            {
             fprintf(stderr,"\nError al alocar memoria dinamica.\n");
             free(pLiberador);
             return ERROR;
            }
           }
        /* Ultimo DNI de la linea */
        sscanf(linea, "%s\n", DNI);
        if ((pAmigo = BuscaDNIPersona(DNI, lista)) != NULL)
          if (!MeterContenido(&(pPersona->amigos), (void *) pAmigo))
           {
            fprintf(stderr,"\nError al alocar memoria dinamica.\n");
            free(pLiberador);
            return ERROR;
           }
       }
     if (fgets(linea, MAX_LENGTH, ptr) == NULL)
       finFichero = SI;
     else
       ErroresLineaDNI(linea);
    } /* Fin: while (finFichero != SI) */
  free(pLiberador);
  fclose(ptr);

 return BIEN;
}

/*********************************************************************
*   Funcion: ESTADO Algoritmo(LISTA *lista)
*
*   IN:
*   OUT:
*   MAKE: Este proceso hace de frontend entre las funciones y el usuario.
*
*********************************************************************/
ESTADO Algoritmo(LISTA *lista)
{
 PERSONA *pPersona;
 char comando, DNI[10] = "", apoyo[10] = "";
 nodo *pNodo;
 LISTA listadeapoyo;


  fprintf(stdout, "\nCommando (h para la ayuda)  ->\t");
  fflush(stdin);
  fgets(&comando, LONGITUD, stdin); /* Leemos la opcion del usuario */
  fprintf(stdout,"\n");

  while ((comando != 'Q') && (comando != 'q'))
   {
    if (*lista == NULL) /* Si ya no existe ningun nodo en la lista. */
     {
```

```
      fprintf(stdout,"\nSe han eliminado todas las personas de la lista.");
      fprintf(stdout,"\nImposible realizar mas operaciones.");
      /* Terminamos porque ya no podemos hacer ninguna otra cosa. */
      return BIEN;
     }
    switch (comando)
     {
      case 'a':
      case 'A':
               listadeapoyo = OrdenarPorNombre(*lista);
               ImprimeLista(listadeapoyo);
               LiberaLista(&listadeapoyo);
               break;

      case 'b':
      case 'B':
               listadeapoyo = OrdenarPorApellido(*lista);
               ImprimeLista(listadeapoyo);
               LiberaLista(&listadeapoyo);
               break;

      case 'c':
      case 'C':
               listadeapoyo = OrdenarPorEdad(*lista);
               ImprimeLista(listadeapoyo);
               LiberaLista(&listadeapoyo);
               break;

      case 'd':
      case 'D':
               listadeapoyo = OrdenarPorDNI(*lista);
               ImprimeLista(listadeapoyo);
               LiberaLista(&listadeapoyo);
               break;

      case 'p':
      case 'e':
      case 'E':
               ImprimirListaAmigos(*lista);
               break;

      case 'f':
      case 'F':
               fprintf(stdout,"\nPor favor introduzca el DNI de la persona a
eliminar->");
               fgets(apoyo, 11, stdin);
               sscanf(apoyo, "%s\n", DNI);
               /* Buscamos a alguien con el DNI dado */
               if ((pPersona = BuscaDNIPersona(DNI, *lista)) == NULL)
                fprintf(stderr,"\nDNI no encontrado, Nadie ha sido
eliminado\n");
               else
                {
                 while((pPersona = BuscaDNIPersona(DNI, *lista)) != NULL)
                  {
                   /* Obtenemos el nodo de la persona */
                   if ((pNodo = BuscaPersona(pPersona, *lista)) == NULL)
                     return ERROR;
                   /* Eliminamos la persona de todos los lugares y nodos */
                   ResolverDependencias(lista, pPersona);
                   if (!EliminaElemento(lista, pNodo))
                     return ERROR;
                   fprintf(stdout,"\nDNI encontrado, se eliminara a %s %s\n",
pPersona->nombre, pPersona->apellido);
                   if (!LiberaPersona((void *) pPersona))
                    {
                     fprintf(stderr, "\nError, no se ha podido eliminar a la
persona.");
```

```
                    return ERROR;
                   }
                 }
               }
              break;
      case 'h':
      case 'H':
               fprintf(stdout,"\nOrdenar por nombre:    'a'");
               fprintf(stdout,"\nOrdenar por apellido:  'b'");
               fprintf(stdout,"\nOrdenar por edad:      'c'");
               fprintf(stdout,"\nOrdenar por DNI:       'd'");
               fprintf(stdout,"\nLista de amigos:       'e'");
               fprintf(stdout,"\nEliminar una persona:  'f'");
               fprintf(stdout,"\nTerminar:              'q'\n");
               break;
      default:
               fprintf(stderr, "\nComando incorrecto.\n");

     } /* Fin: switch (comando) */
    fprintf(stdout, "\nCommando (h para la ayuda)  ->\t");
    fflush(stdin);
    fgets(&comando, LONGITUD, stdin);
    fprintf(stdout,"\n");
   } /* Fin: While... */

 return BIEN;
}

/*****************************************************************
*   Funcion: ESTADO ErroresLineaPersonas(char *linea)
*
*   IN:
*   OUT:
*   MAKE: Este proceso hace de frontend entre las funciones y el usuario.
*
*****************************************************************/
ESTADO ErroresLineaPersonas(char *linea)
{
 char nombre[MAX_LONG], apellido[MAX_LONG], edad[MAX_LONG], dni[MAX_LONG];
 int i=0;

  /* Buscamos todos los errores posibles que pueda haber en la linea */
  sscanf(linea,"%s %s %s %s", nombre, apellido, edad, dni);
  if (strlen(nombre) > MAX_CARACTERES)
    fprintf(stderr,"\nEl nombre %s es demasiado largo.",nombre);
  if (strlen(apellido) > MAX_CARACTERES)
    fprintf(stderr,"\nEl apellido %s es demasiado largo.",apellido);
  if (strlen(edad) > 2)
    fprintf(stderr,"\nLa longugitud de la edad %s debe estar entre 1 y
2",edad);
  ErroresLineaDNI(dni);
  if (edad == "0")
    fprintf(stderr,"\nLa edad de una persona no puede ser 0. Revise el
fichero");
  for (i=0; edad[i] != '\0'; i++)
   {
    if (!isdigit(edad[i]))
      fprintf(stderr,"\nLa edad %s no es correcta.", edad);
   }

 return BIEN;
}

/*****************************************************************
*   Funcion: ESTADO ErroresLineaDNI(char *linea)
*
*   IN:
*   OUT:
```

```
*   MAKE: Este proceso hace de frontend entre las funciones y el usuario.
*
*****************************************************************/
ESTADO ErroresLineaDNI(char *linea)
{
 int i=0, j=0;
 int CharErroneo=0;

  while ((linea[j] != '\n') && (linea[j] != '\x0'))
   {
    for (i=0,j=j; (linea[j] != '\t')&&(linea[j] != '\n')&&(linea[j] != '\x0');
i++,j++)
     {
      if (!isdigit(linea[j]))
     CharErroneo++;
     }
    if (CharErroneo) /* Si se ha encontrado algun caracter erroneo */
     {
      fprintf(stderr,"\nSe han encontrado %d caracteres
erroneos,",CharErroneo);
      fprintf(stderr,"\nse omitiran los DNI que no sean correctos.");
     }
    if ( (i != 7) && (i != 8) )
     {
      fprintf(stderr,"\nEl DNI leido no tiene la longitud adecuada.");
      fprintf(stderr,"\nRevise el fichero para evitar problemas.");
     }
    /* Saltamos el tabulador */
    if (linea[j] == '\t')
       j++;
    CharErroneo = 0;  /* Inicializamos para volver a empezar si es necesrio */
   }

 return BIEN;
}
```

## 6.main.c

```
/*****************************************************************
*   Autor: Victor Sanchez2
*   Web: www.victorsanchez2.net
*   Correo: victorss19@eresmas.com
*   Make: Lee de un archivo de texto, el cual es recibido como un argumento
*   que introduce el usuario, e inserta los datos de las diferentes personas
*         dentro de una lista.
*         El formato del archivo del que se leeran los datos es:
*             nombre\tapellido\t\edad\tDNI\n
*         De un segundo (pasado como argumento siguiente al anterior) archivo
*         se leen los DNI de las personas a las cuales se le asignaran los
*         amifos. El formato es:
*             DNI_Persona\tDNI_Amigo\tDNI_Amigo2...\tDNI_AmigoN\n
*         En primer lugar encontramos el DNI de una persona y luego los DNI de
*         sus amigos.
*         El programa crea una lista en la que podremos ordenar las personas
*         por su nombre, apellido, edad o DNI. Tambien existe una opcion para
*         mostrar los amigos de cada persona y otra para eliminar a alguien
*         escribiendo su DNI.
*
*****************************************************************/

#include "lista.h"
#include "persona.h"
#include "tipos.h"
#include <stdio.h>

void ayuda(void);
```

```
LISTA lista = NULL;

int main(int argc, char *argv[])
{
  /* Comprobamos el numero de parametros */
  if (argc != 3)
   {
    ayuda();
    return ERROR;
   }

  if ((lista = LeerPersonas(argv[1]))==NULL)
   {
    fprintf(stderr,"\nError al leer el archivo de personas, por favor
reviselo.\n");
    LiberarListaPersonas(&lista);
    return ERROR;
   }

  if (!LeerAmigos(argv[2], lista))
   {
    fprintf(stderr,"\nError al leer el archivo de amigos,  por favor
reviselo.\n");
    LiberarListaPersonas(&lista);
    return ERROR;
   }

  if (!Algoritmo(&lista))
   {
    fprintf(stderr,"\nHa ocurrido un error grave, se finalizara el
programa.\n");
    LiberarListaPersonas(&lista);
    return ERROR;
   }

   LiberarListaPersonas(&lista);

 return BIEN;
}

/**********************************************************************
*   Funcion: void ayuda()
*
*   IN:
*   OUT:
*   MAKE: Muestra un mensaje al usuario para que introduzca correctamente
*         los archivos de las personas y los amigos.
*
**********************************************************************/
void ayuda()
{
 fprintf(stderr,"\nUso del programa: <ejecutable> <fichero_personas>
<fichero_amigos>.\n");
}
```

*Estilo de programación: Es especialmente importante el control de errores: es justificable que un programa no admita valores muy grandes de los datos, pero no es justificable que el programa tenga un comportamiento anómalo con dichos valores. Por ejemplo en el ejercicio 3 podría comprobarse que el número N no supere una cantidad*

*dada; lo erróneo sería que el usuario pusiese por ejemplo N=100000 y el programa "cascase" o se quedase "colgado".*

Esto es todo por ahora, pronto intentaré poner más ejercicios (en la medida que mis estudios me lo permitan) . Sólo me gustaría pediros una cosa, si distribuís estos ejercicios entre vuestros amigos o a través de Internet, por favor, dejar mi nombre, me ha costado un gran trabajo.

Está permitida su distribución para fines educativos, nunca lucrativos.

Por favor, escriban ante cualquier duda, comentario o errata que sirva para mejorar los ejercicios.

*¡¡¡ Ánimo !!!*