UNIVERSIDAD DE MÁLAGA | uma.es

**Dpto. de Lenguajes y Ciencias de la Computación**

# Programación de Sistemas y Concurrencia

Control Bloque 1 – Temas 1 y 2

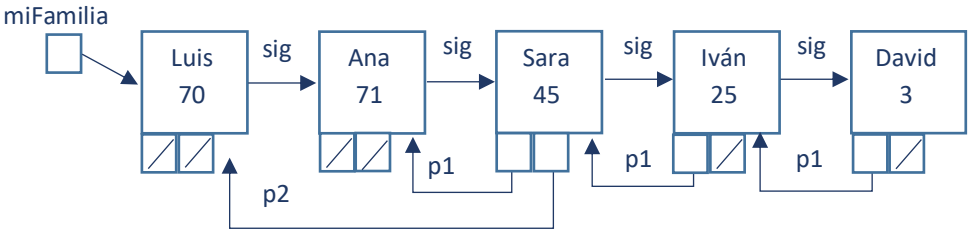
Curso 2019-2020

APELLIDOS _____ NOMBRE _____

DNI _____ ORDENADOR _____ GRUPO/TITULACIÓN _____

## Parte 1 (obligatoria para aprobar el examen)

En este ejercicio vamos diseñar un sistema que gestiona la información de los miembros de una familia. Para ello el sistema utiliza una lista enlazada que almacena información básica de los miembros: nombre, edad, y la relación de parentesco con sus progenitores si estos forman parte de la familia. A continuación se muestra un ejemplo de esta estructura:




Donde sig nos indica cual es el siguiente miembro de la familia (en el orden en el que se insertó), y p1 y p2 indican cual es el progenitor 1 y 2 respectivamente. Una persona tiene como máximo dos progenitores. En el ejemplo Luis y Ana no tienen progenitores en la familia, Sara tiene dos progenitores, e Iván y David solo tienen un progenitor. Una familia tiene como mínimo 1 miembro, pero no tiene limitado el número máximo de miembros. Para simplificar, asumimos que todos los miembros de una familia tienen nombres diferentes y que los nombres no son compuestos.

Para implementar esta estructura y los métodos necesarios para su manipulación, vamos a utilizar la siguiente definición de tipos, que se encuentra en el fichero Familia.h:

```
#define MAX_NOMBRE 20 //longitud máxima de los nombres

typedef struct Persona* PtrPersona;

struct Persona{
        char nombre[MAX_NOMBRE];
        int edad;
        PtrPersona p1; //progenitor1
        PtrPersona p2; //progenitor2
        PtrPersona sig;

};
```

Utilizando estos tipos, hay que implementar las siguientes funciones en el fichero Familia.c. Las funciones deben mostrar mensajes de error/aviso en caso de que se produzca algún error o incidencia:

```
/* Crea una familia con los datos que un usuario introduce por teclado:
 *  - nombre de la persona: si el nombre es "*" se da por finalizada la lectura de datos.
```

```
 *  - edad
 *  - nombre del progenitor 1/2: si el nombre es "-" o no se encuentra aún ese nombre en
 *                               la familia, se asume que no tiene progenitor 1 o 2.
 */
PtrPersona crearFamilia();

/* Busca en la familia una persona con el nombre dado.
 * Devuelve una referencia a los datos de la persona en la lista.
 * Si la persona no forma parte de la familia, esta referencia valdrá NULL.
 */
void buscarPersona (PtrPersona familia, const char *nombre, PtrPersona *persona);

/* Muestra por pantalla la información de los miembros de la familia.
 * Para cada persona se muestra nombre, edad, y nombre de los progenitores si existen.
 */
void mostrarFamilia(PtrPersona familia);

/* Elimina todos los miembros de una familia, liberando correctamente la memoria
 * de la estructura.
 */
void destruirFamilia(PtrPersona *familia);
```

Podéis usar el fichero Driver1.c para comprobar el correcto funcionamiento de las funciones anteriores.

## Parte 2: permite llegar al notable. Sólo se evalúa si se ha aprobado

Implementar las siguientes funciones en el fichero Familia.c resultante de la parte 1. Para comprobar el correcto funcionamiento podéis usar el fichero Driver2.c.

```
/* Elimina a la persona cuyo nombre coincide con el que se pasa como parámetro.
 * Si no hay una persona con ese nombre, la familia no se modifica.
 * Si existe una persona con ese nombre, además de eliminarlo de la lista, actualiza
 * correctamente los progenitores del resto de personas que siguen en la familia.
 * ok almacena si se ha borrado la persona de la familia.
 */
void eliminarPersona(PtrPersona *familia, char *nombre, int *ok);

/* Guarda en un fichero de texto la información de la familia. El alumno elige el
 * formato en el que se almacena dicha información.
 * Esta función devuelve el número de personas que se han guardado en el fichero.
 */
int guardarFicheroTexto(char *nombreFichero, PtrPersona familia);

/* Guarda en un fichero binario la información de la familia. El alumno elige el
 * formato en el que se almacena dicha información, pero debe ser compatible con
 * el formato utilizado en la función cargarFicheroBinario.
 * Esta función devuelve el número de personas que se han guardado en el fichero.
 */
int guardarFicheroBinario(char *nombreFichero, const PtrPersona *familia);
```

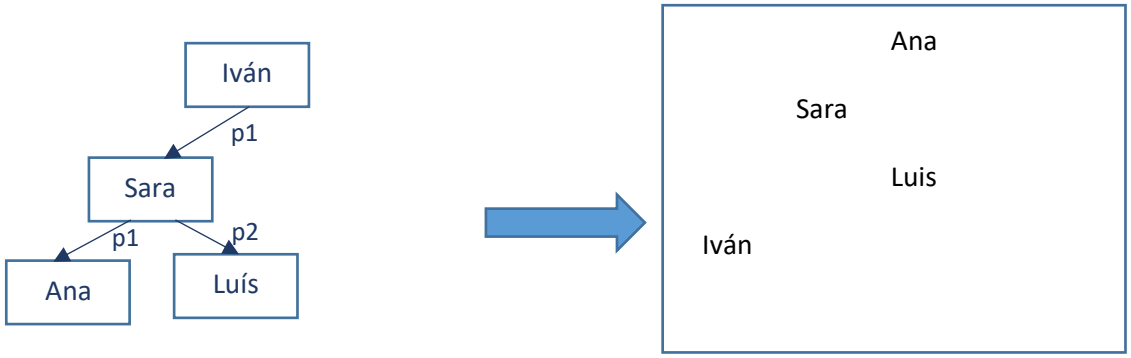
## Parte 3: Permite llegar a sobresaliente/matrícula. Sólo se evalúa si se ha llegado al notable

Además de implementar las siguientes funciones, hay que implementar un método principal (fichero Driver3.c) que demuestre que la parte 2 funciona correctamente.

```
/* Crea una nueva familia utilizando la información leída de un fichero binario.
 * El fichero almacena la información con el formato que produce la función guardarFicheroBinario.
 * Esta función devuelve en nPersoas el número de personas que se han guardado en el fichero.
 * /
PtrPersona cargarFicheroBinario(char *nombreFichero, int *nPersonas);
```

```
/* Muestra por pantalla el árbol de ascendentes de la persona con el nombre dado.
 * Si no hay personas con ese nombre, se muestra un mensaje.
 * Si hay una persona con ese nombre, la raíz de dicho árbol será esta persona.
 */
void mostrarArbolAscendentes(PtrPersona familia, char *nombre);
```

Para la familia que aparece el ejemplo inicial, el árbol de ascendentes de Iván es el que se muestra a la izquierda. Para obtener el árbol se valorará el empleo de la recursividad en la implementación. La llamada mostarArbolAscendentes(miFamilia, "Iván") debe mostrar el formato que aparece a la derecha:




**Anexo.** Los prototipos de las funciones para manipular strings (están en <string.h>) son:

**char* strcpy(char *s1, char *s2)**: Copia los caracteres de la cadena s2 (hasta el carácter '\0', incluido) en la cadena s1. El valor devuelto es la cadena s1.

**int strcmp(char *s1, char *s2)**: Devuelve 0 si las dos cadenas son iguales, <0 si s1 es menor que s2, y >0 si s1 es mayor que s2.

**size_t strlen(const char *s)**: Calcula el número de caracteres de la cadena apuntada por s. *Esta función no cuenta el carácter '\0' que finaliza la cadena*.

Los prototipos de las funciones de lectura y escritura en ficheros de la biblioteca <stdio.h> son los siguientes (se dan por conocidos los prototipos de las funciones de <stdlib.h> que necesites, como free o malloc):

**FILE *fopen(const char *path, const char *mode):** Abre el fichero especificado en el modo indicado ("rb"/ y "wb" para lectura/escritura binaria y "rt"/"wt" para lectura/escritura de texto). Devuelve un puntero al manejador del fichero en caso de éxito y NULL en caso de error.

**int fclose(FILE *fp):** Guarda el contenido del buffer y cierra el fichero especificado. Devuelve 0 en caso de éxito y -1 en caso de error.

*LECTURA / ESCRITURA BINARIA*
**unsigned fread(void *ptr, unsigned size, unsigned nmemb, FILE *stream):** Lee nmemb elementos de datos, cada uno de tamaño size bytes, desde el fichero stream, y los almacena en la dirección apuntada por ptr. Devuelve el número de elementos leídos.

**unsigned fwrite(const void *ptr, unsigned size, unsigned nmemb, FILE *stream):** Escribe nmemb elementos de datos, cada uno de tamaño size, al fichero stream, obteniéndolos desde la dirección apuntada por ptr. Devuelve el número de elementos escritos.
*LECTURA/ESCRITURA TEXTO*
**int fscanf(FILE *stream, const char *format, ...):** Lee del fichero stream los datos con el formato especificado en el parámetro format, el resto de parámetros son las variables en las que se almacenan los

datos leídos en el formato correspondiente. La función devuelve el número de variables que se han leído con éxito.

**int fprintf(FILE *stream, const char *format, …)**: Escribe en el fichero stream los datos con el formato especificado en el parámetro format. El resto de parámetros son las variables en las que se almacenan los datos que hay que escribir. La función devuelve el número de variables que se han escrito con éxito.