Guía para la realización de la Práctica 3 – Tema 2

Programación de Sistemas y Concurrencia (2019-2020)

Pasos a seguir

La práctica 3 es la última práctica del bloque de programación de sistemas de la asignatura. Para realizarla os recomendamos seguir los siguientes pasos:

- Descargar y leer el enunciado.
- Descargar el material proporcionado: la plantilla de la práctica (decrypt.c) y algunos ejemplos de ficheros para descodificar (*.enc)
- Seguir las instrucciones que aparecen en esta guía
- Enviar las dudas que tengáis al foro de la práctica o apuntarse a tutorías virtuales

Introducción

La práctica consiste en implementar un programa capaz de descifrar una imagen codificada. En la práctica vamos a trabajar principalmente con:

- **Ficheros binarios**
  - El fichero con la imagen encriptada lo abriremos en modo lectura binario
    - crisantemo.png.enc, imagen.enc e imgno8.bmp.enc
  - Las imágenes descifradas las almacenaremos en ficheros abiertos en modo escritura binario
    - crisantemo.png, imagen.png y imgno8.bmp
- **Operaciones de bajo nivel** para implementar el algoritmo de descifrado

Introducción

Podemos dividir la práctica en dos partes:

- **Parte1: Implementación del algoritmo de descifrado.**
  - Implementaremos una función que dado un contenido codificado (en un buffer) y una clave, aplica el algoritmo, devolviendo una versión decodificada del contenido (buffer).
- **Parte2: Uso del algoritmo para descrifrar la imagen almacenada en un fichero**
  - Implementaremos el método main para que dado el nombre de fichero:
    - Lea el contenido del fichero y lo almacene en un buffer
    - Ejecute el algoritmo de descifrado con los datos del buffer
    - Almacene los datos descifrados en otro fichero

Parte 1 – Algoritmo de descifrado

- El algoritmo de descifrado se describe en el enunciado

Para cada bloque de 64 bits (`unsigned int v[2]`), sea la clave *k* (`unsigned int k[4]`), y *delta* una constante igual a 0x9e3779b9:

Inicializar *sum* a 0xC6EF3720

Repetir 32 veces:

Restar a `v[1]` la aplicación del operador XOR (^) a
(`v[0]` desplazado a la izquierda 4 bits +`k[2]`),
(`v[0]` + `sum`) y
(`v[0]` desplazado a la derecha 5 bits)+`k[3]`

Restar a `v[0]` la aplicación del operador XOR (^) a:
(`v[1]` desplazado a la izquierda 4 bits + `k[0]`)
(`v[1]`+ `sum`) y
(`v[1]` desplazado a la derecha 5 bits)+`k[1]`

Restar a `sum` el valor de `delta`.

Al final de las 32 iteraciones, se tendrá en `v` el valor desencriptado de los 64 bits.

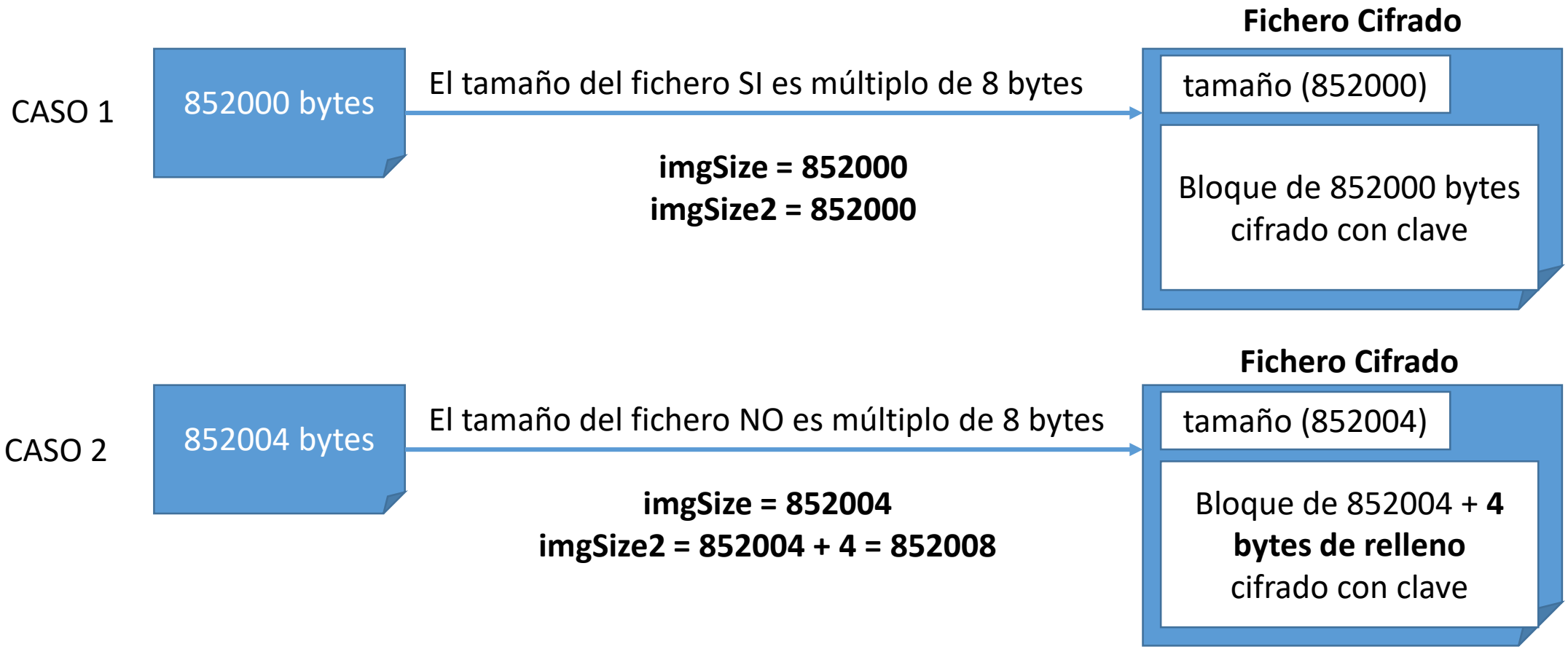
Operaciones de bajo nivel:
- XOR
- Desplazamiento a la izquierda
- Desplazamiento a la derecha

Parte 1 – Algoritmo de descifrado

- Se implementa en el siguiente método del fichero decrypt.c, que dado un bloque encriptado de 64 bits (parámetro v) y la clave de 128 bits (parámetro k) lo descifra y el resultado lo guarda en v.

```
/* Parte 1: algoritmo de descifrado
 * v: puntero a un bloque de 64 bits.
 * k: puntero a la clave para descifrar.
 * Sabiendd que unsiged int equivale a 4 bytes (32 bits)
 * Podemos usar la notación de array con v y k
 * v[0] v[1] --- k[0] ... k[3]
 */
void decrypt(unsigned int *v, unsigned int *k)
{
   //Definir variables e inicializar los valores de delta y sum

   //Repetir 32  veces (usar un bucle) la siguiente secuencia de operaciones de bajo nivel

         //Restar a v[1] el resultado de la operacion :
            // (v[0] desplazado a la izquierda 4 bits +k[2]) XOR (v[0] + sum)  XOR (v[0] desplazado a la derecha 5 bits)+k[3]

         //Restar a v[0] el resultado de la operacion:
            // (v[1] desplazado a la izquierda 4 bits + k[0]) XOR (v[1]+ sum)  XOR (v[1] desplazado a la derecha 5 bits)+k[1]

         // Restar a sum el valor de delta
}
```

Parte 2: Descifrado de un fichero



Parte 2: Descifrado de un fichero

- Desde la función main() se llamará a la función decrypt() anterior para ir descifrando los datos del fichero en bloques de 64 bits

```
/* Parte 2: Metodo main. Tenemos diferentes opciones para obtener el nombre del fichero cifrado y el descifrado
 * 1. Usar los argumentos de entrada (argv)
 * 2. Pedir que el usuario introduzca los nombres por teclado
 * 3. Definir arrays de caracteres con los nombres
 */
int main(int argc, char *argv[] )
{
    /*Declaración de las variables necesarias, por ejemplo:
    * variables para los descriptores de los ficheros ( FILE * fent, *fsal)
    * la constante k inicializada con los valores de la clave
    * buffer para almacenar los datos (puntero a unsigned int, más adelante se reserva memoria dinámica */

    /*Abrir fichero encriptado fent en modo lectura binario
     * nota: comprobar que se ha abierto correctamente*/


    /*Abrir/crear fichero fsal en modo escritura binario
     * nota: comprobar que se ha abierto correctamente*/
```

- Continua en la siguiente transparencia …

Parte 2: Descifrado de un fichero

Continua de la siguiente transparencia …

```
/*Al comienzo del fichero cifrado esta almacenado el tamaño en bytes que tendrá el fichero descifrado.
 * Leer este valor (imgSize)*/

 /*Reservar memoria dinámica para el buffer que almacenara el contenido del fichero cifrado
  * nota1: si el tamaño del fichero descifrado (imgSize) no es múltiplo de 8 bytes,
  * el fichero cifrado tiene además un bloque de 8 bytes incompleto, por lo que puede que no coincida con imgSize
  * nota2: al reservar memoria dinámica comprobar que se realizó de forma correcta */

 /*Leer la información del fichero cifrado, almacenado el contenido en el buffer*/

 /*Para cada bloque de 64 bits (8 bytes o dos unsigned int) del buffer, ejecutar el algoritmo de desencriptado*/

 /*Guardar el contenido del buffer en el fichero fsal
  * nota: en fsal solo se almacenan tantos bytes como diga imgSize */

 /*Cerrar los ficheros*/
}
```

Parte 2: Descifrado de un fichero

AYUDA adicional:

- Codificación de la clave proporcionada en el enunciado:
  unsigned int v[2], k[4] = {128, 129, 130, 131};

- El tamaño del fichero (primer dato leído del fichero y que será almacenado en **imgSize**) puede ser múltiplo de 8 o no. Si no lo es hay que cambiar el tamaño a un valor múltiplo de ocho (guardado en **imgSize2**)

```
if (imgSize % 8 != 0) {     //El tamaño no es divisible por 8
    //Cambiamos el valor de imgSize2 para poder leer relleno
    imgSize2 = imgSize + (8 - (imgSize % 8));
}else{
    imgSize2 = imgSize;
}
```

Parte 2: Decodificación de un fichero

AYUDA adicional:

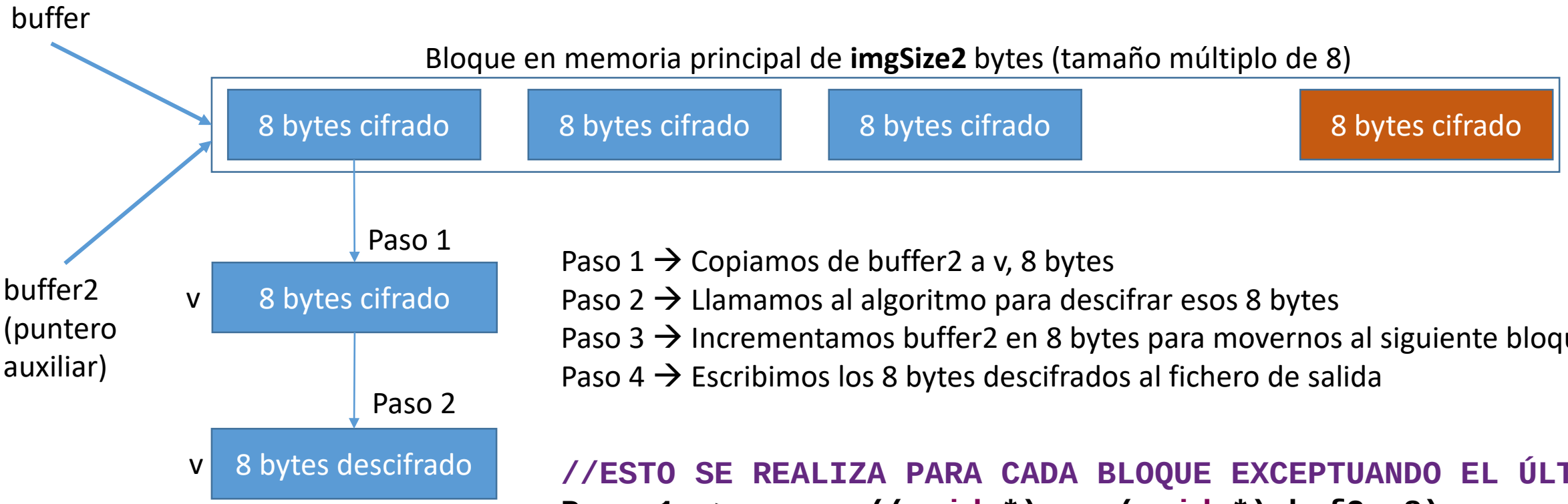
- Tras reservar memoria dinámica para almacenar el contenido de la imagen a descifrar, realizamos una sola operación de lectura para cargar el contenido del fichero a memoria dinámica:

- Leemos **imgSize2** valores de tamaño char (1 byte) y los guardamos en **buffer**

```
leidos = fread((void *) buffer, sizeof(char), imgSize2, fCif);
if (leidos != imgSize2) {
        //Si no se han podido leer bien no seguimos
        //Cerramos los ficheros y liberamos la memoria dinámica
}
```

Parte 2: Decodificación de un fichero

AYUDA adicional:

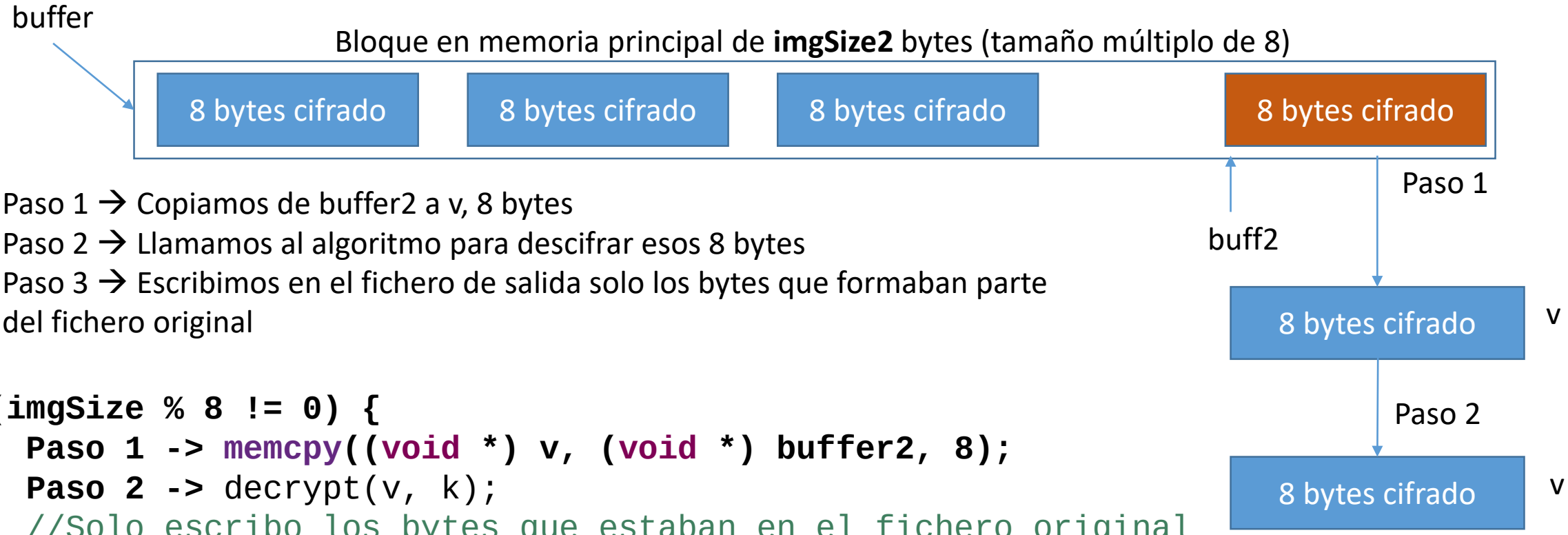


Paso 1 → Copiamos de buffer2 a v, 8 bytes
Paso 2 → Llamamos al algoritmo para descifrar esos 8 bytes
Paso 3 → Incrementamos buffer2 en 8 bytes para movernos al siguiente bloque
Paso 4 → Escribimos los 8 bytes descifrados al fichero de salida

```
//ESTO SE REALIZA PARA CADA BLOQUE EXCEPTUANDO EL ÚLTIMO
Paso 1 -> memcpy((void *) v, (void *) buf2, 8);
Paso 2 -> decrypt(v, k);
Paso 3 -> buf2 += 8;
Paso 4 -> fwrite((void *) v, sizeof(char), 8, fsal);
```

Parte 2: Decodificación de un fichero

AYUDA adicional: El último bloque se gestiona diferente porque podría tener bytes de relleno



Paso 1 → Copiamos de buffer2 a v, 8 bytes
Paso 2 → Llamamos al algoritmo para descifrar esos 8 bytes
Paso 3 → Escribimos en el fichero de salida solo los bytes que formaban parte del fichero original

```
if (imgSize % 8 != 0) {
     Paso 1 -> memcpy((void *) v, (void *) buffer2, 8);
     Paso 2 -> decrypt(v, k);
     //Solo escribo los bytes que estaban en el fichero original
     Paso 3 -> fwrite((void *) v, sizeof(char), imgSize % 8, fsal);
}
```

Parte 2: Decodificación de un fichero

AYUDA adicional:

- Se puede usar la siguiente función para copiar un número de bytes de un origen a un destino.

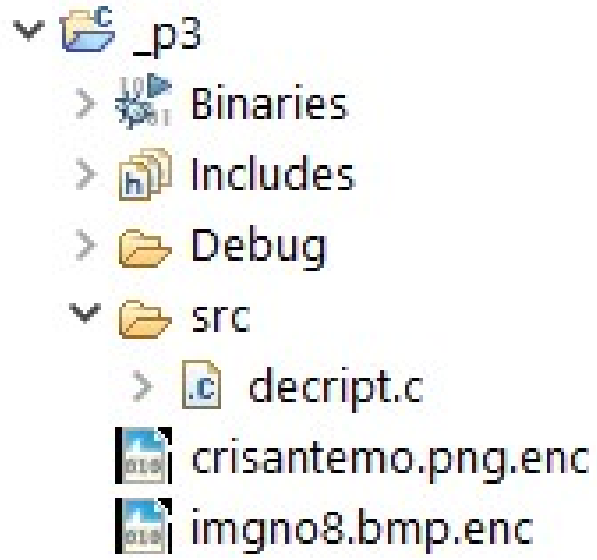
void *memcpy(void *dest, const void * orig, size_t n);

- **Parámetros:**
  - **dest** – Puntero a la zona de memoria destino donde los datos serán copiados.
  - **orig** – Puntero a la zona de memoria donde están los datos que se quieren copiar.
  - **n** – Número de bytes que se copiarán de origen a destino.

- **Valor de retorno:**
  - Puntero a la zona de memoria destino.

¿Cómo probamos nuestro programa?

- Si usamos eclipse, hay que copiar los ficheros encriptados en la carpeta principal de nuestro proyecto



- Si usamos compilación por línea de comandos, copiamos los ficheros encriptados en la misma carpeta donde esté el ejecutable