En esta práctica se utilizarán strings, ficheros binarios, estructuras híbridas (union), estructuras con campos de bits, operaciones con bits, compilación por separado, argumentos en el main, y el *símbolo del sistema*.

Consiste en crear un programa que encripte o desencripte cualquier fichero, binario o no. La **encriptación se realiza en dos niveles**: primeramente, se aplica la encriptación usada en la práctica anterior; seguidamente, se aplica una segunda encriptación, invirtiendo determinados bits. La **desencriptación** se realiza **en sentido inverso**: primeramente, se desinvierten los bits; seguidamente, se aplica la desencriptación usada en la práctica anterior. Tras la encriptación se elimina el fichero original, creándose otro con la extensión ".CODED". Tras la desencriptación se elimina el fichero ".CODED" y se crea otro con la extensión ".DECODED".

El programa, denominado **criptex**, que se ejecutará desde el *símbolo del sistema*, consta de los siguientes parámetros:

- primer parámetro: opción "–c" para encriptar o "-d" para desencriptar
- segundo parámetro: bits a invertir (string de ocho chars)
- tercer parámetro: fichero a encriptar o desencriptar

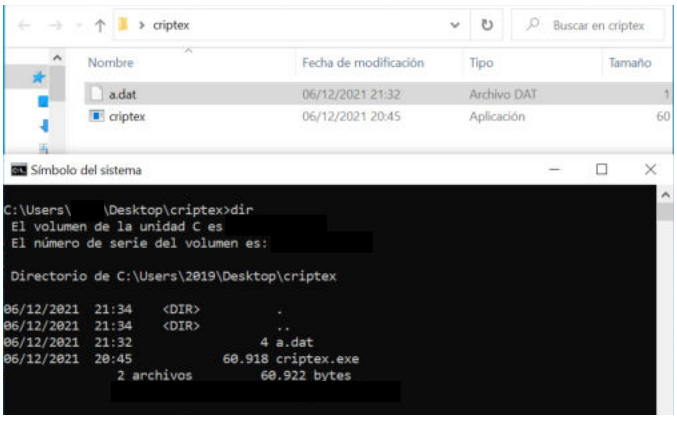
Se deberá **comprobar** que se introducen éstos tres parámetros y **que son correctos**.

Para realizar el programa, se deberán implementar las siguientes funciones:

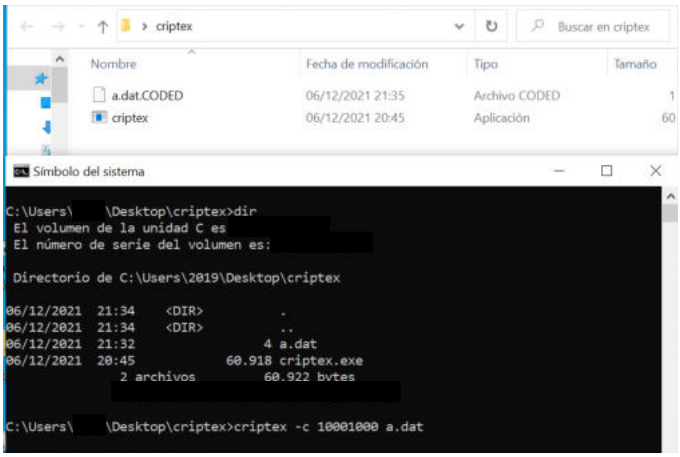
- **void rotar_char(unsigned char *c)** que encripta un char con la encriptación usada en la práctica anterior: rota los dos bits de más a la derecha, pasando a ser los de más a la izquierda, y desplazando el resto de bits hacia la derecha.
- **void desrotar_char(unsigned char *c)** que desencripta un char encriptado con rotar_char().
- **void invertir_bits_char (unsigned char *c , char * bits_invertir)** que invierte los bits de un char especificados por bits_invertir.

Estas funciones se implementarán en un fichero .c de compilación por separado, y además se implementará el correspondiente fichero .h en el que se incluirán los prototipos de las funciones descritas y los tipos que se definan.

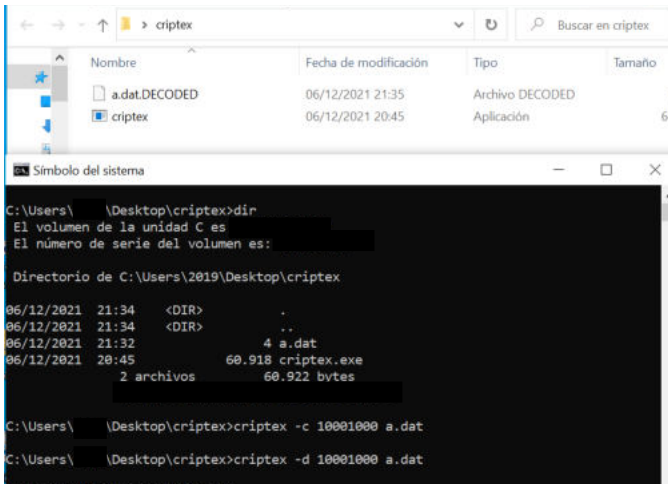
Un ejemplo completo de funcionamiento, se muestra en las capturas de pantalla siguientes:



Paso 1: Se ubica el ejecutable y el binario en una carpeta



Paso 2: Se encripta, rotando los dos bits de menos peso e invirtiendo los bits 10001000



Paso 3: Se desencripta, desinvirtiendo los bits 10001000 y desrotando

Comprobar que **criptex** es capaz de encriptar / desencriptar un fichero binario, un fichero de texto y un fichero ejecutable.

Solución:

Fichero .h

```h
#ifndef CIPHER_H
    #define CIPHER_H

    char* cph_crot    (unsigned char*);
    char* cph_cdesrot (unsigned char*);
    char* cph_cinvert (unsigned char*, unsigned char);
    void  cph_fencode (FILE*, FILE*, unsigned char);
    void  cph_fdecode (FILE*, FILE*, unsigned char);

#endif // CIPHER
```

Fichero .c (de funciones)

```c
#include <stdio.h>
#include "cipher.h"

char* cph_crot (unsigned char* c)
{
```

```
    *c = ( ( *c & 3 ) << 6 ) | (*c >> 2);
    return c;
}

char* cph_cdesrot (unsigned char* c)
{
    *c = ( ( *c & 192 ) >> 6 ) | (*c << 2);
    return c;
}
char* cph_cinvert ( unsigned char* c, unsigned char byte)
{
    *c ^= byte;
    return c;
}

void cph_fencode (FILE* f_in, FILE* f_out, unsigned char byte)
{
    unsigned char c;
    while((c = fgetc(f_in)) && !feof(f_in))
        fputc(*cph_cinvert(cph_crot(&c), byte), f_out);
}

void cph_fdecode (FILE* f_in, FILE* f_out, unsigned char byte)
{
    unsigned char c;
    while((c = fgetc(f_in)) && !feof(f_in))
        fputc(*cph_cdesrot(cph_cinvert(&c, byte)), f_out);
}
```

Fichero .c (del main)

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

#define OPT_NONE -1
#define OPT_C 0
#define OPT_D 1

#define EXTENSION_DECODE "DECODED"
#define EXTENSION_CODE   "CODED"

#define NAME_CAP    90
#define NAME_NEWCAP 100
```

```
int           isByte        (char*);
void          fextension    (char*, char*, char*);
unsigned char cbinToDecimal (char[8]);

int main(int argc, char* argv[])
{
    FILE         *f_in, *f_out;
    int           opt = OPT_NONE;
    unsigned char byteValue;
    char          fnewName[NAME_NEWCAP];

    if (argc != 4)
    {
        printf("\nERROR: Faltan/Sobran argumentos");
        printf("\n       [ Argumentos: -c/-d Byte('0'/'1')/(0-255) Filename ]\n");
        return 1;
    }

    if (!strcmp(argv[1], "-c"))
        opt = OPT_C;
    else if (!strcmp(argv[1], "-d"))
        opt = OPT_D;
    else
    {
        printf("\nERROR: Argumento 1: Opcion no reconocida");
        printf("\n       [ Argumentos: -c/-d Byte('0'/'1')/(0-255) Filename ]\n");
        return 2;
    }

    if (strlen(argv[2]) == 8)
        byteValue = cbinToDecimal(argv[2]);
    else if (atoi(argv[2]) >= 0 && atoi(argv[2]) <= 255)
        byteValue = atoi(argv[2]);
    else
    {
        printf("\nERROR: Argumento 2: Tiene que tener 8 char ('0'/'1') o estar entre 0-
255");
        printf("\n       [ Argumentos: -c/-d Byte('0'/'1')/(0-255) Filename ]\n");
        return 3;
    }

    if (!isByte(argv[2]))
    {
        printf("\nERROR: Argumento 2: Tiene que ser compuesto de '0' y '1'");
        printf("\n       [ Argumentos: -c/-d Byte('0'/'1')/(0-255) Filename ]\n");
        return 4;
    }

    if (strlen(argv[3]) > NAME_CAP)
    {
        printf("\nERROR: El nombre del fichero es demasiado grande\n");
        return 5;
```

```
    }

    if (!(f_in = fopen(argv[3], "r")))
    {
        printf("\nERROR: No se pudo abrir el archivo %s\n", argv[3]);
        return 5;
    }

    switch (opt)
    {
        case OPT_C:
            fextension(fnewName, argv[3], EXTENSION_CODE);
            if (!(f_out = fopen(fnewName, "w")))
            {
                printf("\nERROR: No se pudo abrir el archivo %s\n", fnewName);
                return 5;
            }
            cph_fencode(f_in, f_out, byteValue);
            break;

        case OPT_D:
            fextension(fnewName, argv[3], EXTENSION_DECODE);
            if (!(f_out = fopen(fnewName, "w")))
            {
                printf("\nERROR: No se pudo abrir el archivo %s\n", fnewName);
                return 5;
            }
            cph_fdecode(f_in, f_out, byteValue);
            break;
    }

    fclose (f_in);
    fclose (f_out);

    return 0;
}

int isByte(char* byte)
{
    for (; *byte && (*byte == '1' || *byte == '0'); byte++)
    return *byte == '1' || *byte == '0';
}

void fextension(char* fnewName, char* fname, char* fextension)
{
    char* i = strchr(fname, '.');
    if (!i)
    {
        strcpy(fnewName, fname);
        strcat(fnewName, ".");
    }
    else
```

```
        strncpy(fnewName, fname, i - fname + 1);
    strcat(fnewName, fextension);
}

unsigned char cbinToDecimal(char bits[8])
{
    return
        (bits[0] == '1') << 7 +
        (bits[1] == '1') << 6 +
        (bits[2] == '1') << 5 +
        (bits[3] == '1') << 4 +
        (bits[4] == '1') << 3 +
        (bits[5] == '1') << 2 +
        (bits[6] == '1') << 1 +
        (bits[7] == '1');
}
```