IC2001 Estructuras de Datos

Grupo 2

Jose Pablo Aguero Mora 2021126372

Katerine Guzmán Flores 2019390523

Manual Técnico

Proyecto 3B

I Semestre 2022

Tabla de Contenidos

Descripción:

Este proyecto consiste en un encriptador/respaldador de seguridad y un desencriptador/restaurador de archivos originales. Utiliza manejo de bits para crear los diferentes método. El programa posee 5 métodos o tipos de encriptación, con sus 5 métodos de desencriptación. Los cuales serán ejecutados según la decisión del usuario mediante la linea de comandos con las 5 opciones disponibles. Para verificar que efectivamente se encriptaron los archivos o se desencriptaron, se puede verificar ambos procesos desde la carpeta donde se encuentra el ejecutable del proyecto. Se puede encriptar y desencriptar cualquier tipo de archivo y se restaurará con el mismo peso y funcionalidad que el archivo original.

Función main:

Dentro de esta función se crean dos variables de tipo archivo para que sea el archivo de origen de donde se leen los datos y el archivo destino donde se van a escribir los bytes resultantes. Se crea un array de char para ir metiendo lo que se va leyendo del archivo. Hay que recordar que en consola el primer argumento es el nombre del programa, es decir, el argumento 0, porque lo que se utiliza el argumento 1 para saber cual opción se desea, “e” para encriptar y “d”, para desencriptar. Posteriomente se lee el argumento que corresponde a el tipo o método ya sea de encriptación o desencriptación deseado del 1 al 5. El argumento en la posición 3 corresponde al archivo fuente y el argumento 4 corresponde al archivo destino.

```
// Archivos logicos : Buffers tipo FILE
// Se crean dos archivos
FILE* ArchivoOrigen, * ArchivoDestino;

//-----------------------------------------------------------------
// APERTURA DE ARCHIVO FUENTE Y DESTINO
/* Apertura del archivo original, para lectura en binario*/
fopen_s(&ArchivoOrigen, argv[3], "rb");
if (ArchivoOrigen == NULL)
{
    perror("No se puede abrir archivo origen ");//, argv[2]
    return -1;
}

/* Apertura del archivo destino, para escritura en binario*/
fopen_s(&ArchivoDestino, argv[4], "wb");
if (ArchivoDestino == NULL)
{
    perror("No se puede abrir archivo destino"); // , argv[2]
    return -1;
}
```

Encriptar

En caso de que la opción digitada sea de encriptar, se crea una variable de tipo entero que va leyendo los bits del archivo fuente. Mientras se haya leído algo del archivo, se entra a un switch donde se encuentran los tipos de encriptación. Las opciones que existen corresponden a:

Caso 1: Veneno correspondiente a Invertir bit

Caso 2: Veneno correspondiente a Encender bit

Caso 3: Veneno correspondiente a Apagar bit

Caso 4: Veneno correspondiente a Apagar Bit Segundo y Cuarto

Caso 5: Veneno correspondiente a Invertir bit en posicion 4 y enciende bit en posicion 5

```
while (leidos > 0) {
    switch (tipo) {
        // ********************************************** VENENOS **********************************************
    case '1': // invertir bit
        Veneno_1(buffer, leidos); // Ojo esta usando el mismo metodo de desencriptar ya que solo INVIERTE el Bit 0
        break;

    case '2':
        Veneno_2(buffer, leidos); // encender bit
        break;

    case '3':
        Veneno_3(buffer, leidos); // apagar bit
        break;


    case '4':
        Veneno_4(buffer, leidos); // ApagarBit Segundo y Cuartos
        break;

    case '5':
        Veneno_5(buffer, leidos); // Invierte bit en posicion 4 y enciende bit en posicion 5

        break;
```

Una vez que se salga del switch y haya aplicado el veneno, se van metiendo esos bits en el archivo de destino y se pasa al siguiente bloque de 1000 elementos de tipo char.

Desencriptar:

En caso de que se digite cualquier otra letra que no sea “e” entrará en el ciclo muy similar al de encriptar. De igual manera, se tiene una variable leídos que corresponde a lo que se vaya leyendo del archivo fuente. Posteriormente, se entra en un switch para elegir el tipo de restauración correspondiente con el veneno aplicado. Por ejemplo, si se ha aplicado el veneno 2 a un archivo este se debe desencriptar con el antídoto correspondiente. Que de igual forma es el veneno aplicado de forma inversa. De igual manera, se van metiendo los bytes en el archivo destino y se lee el siguente bloque de bytes si existen más. Las opciones que existen corresponden a:

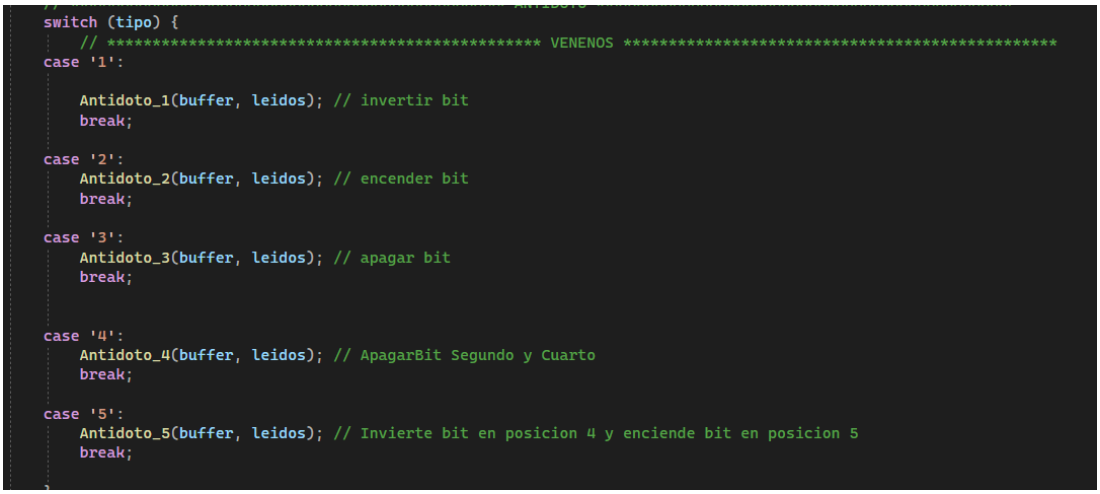
Caso 1: Antídoto correspondiente a veneno de Invertir bit

Caso 2: Antídoto correspondiente a veneno de Encender bit

Caso 3: Antídoto correspondiente a veneno de Apagar bit

Caso 4: Antídoto correspondiente a veneno de Apagar Bit Segundo y Cuarto

Caso 5: Antídoto correspondiente a veneno de Invertir bit en posicion 4 y enciende bit en posicion 5



```
switch (tipo) {
    // ********************************************* VENENOS *********************************************
case '1':

    Antidoto_1(buffer, leidos); // invertir bit
    break;

case '2':
    Antidoto_2(buffer, leidos); // encender bit
    break;

case '3':
    Antidoto_3(buffer, leidos); // apagar bit
    break;


case '4':
    Antidoto_4(buffer, leidos); // ApagarBit Segundo y Cuarto
    break;

case '5':
    Antidoto_5(buffer, leidos); // Invierte bit en posicion 4 y enciende bit en posicion 5
    break;
```

Al terminar ya sea encriptar o desencriptar, en ambos casos se cierran los archivos fuente y destino.

```
fclose(ArchivoOrigen);
fclose(ArchivoDestino);
```

Venenos

Para este algoritmo de manejo de bits como para todos los demás, se utilizan 3 términos o variables, que corresponden a las siguientes:

int & Simbolo: Es el conjunto de 8 bits en donde se realizara el cambio.

Int cual: Corresponde a la posición específica del cambio de bit.

Mascara: Es en conjunto de 8 bits que se utilizan con las compuertas lógicas especificas para cada encriptación.

Caso 1:

En el caso 1 se aplica invertir bit en la posición deseada según el veneno. Primero se enciende el bit cero de la máscara, luego se aplica el corrimiento según la posicion deseada y posteriormente se ejecuta un xor para invertir el bit.

El veneno consiste en tomar el bloque inicial de 1000 bytes e ir recorriendo uno por uno de esos bits en la función de invertir bit para aplicarle el debido proceso. Se utiliza la función, atoi para poder introducir el parámetro como entero en invertir bit.

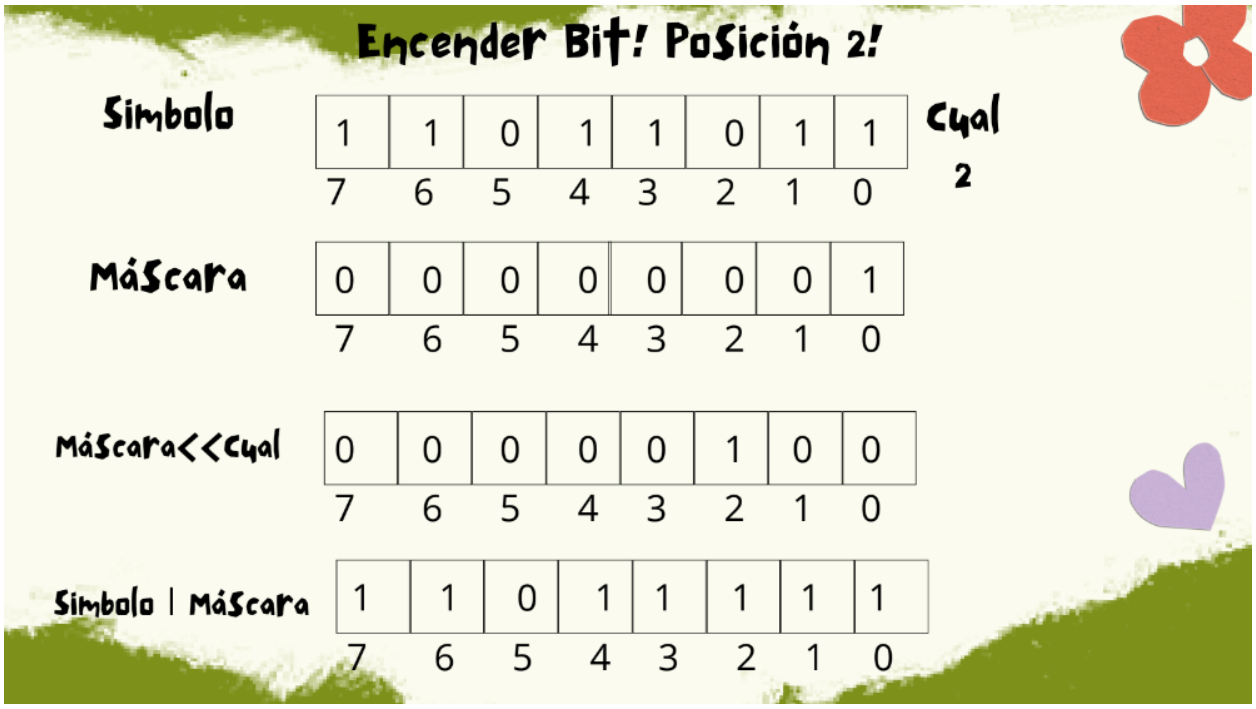
```
// ************************************************** InvertirBit **************************************************
// Coloca 0 si el bit es 1 y viceversa, en la posicion deseada
void InvertirBit(int& Simbolo, int cual)
{
    int Mascara = 1; // se pone el bit encendido

    Mascara = Mascara << cual; // Se coloca el bit encendido en la posicion cual
    Simbolo = Simbolo ^ Mascara; // Se procede a hacer un xor con la mascara y el byte inicial
}

// Se aplica el veneno correspondiente
void Veneno_1(char Bloque[1000], int limite)
{
    int i;
    int posicion_a_invertir = 0; // Se asigna la ubicacion del bit a invertir

    for (i = 0; i <= limite; i++) { // se hace un ciclo para ir introduciendo bit a bit los char del bloque
        int numero = atoi(Bloque); // se pasa el char que contiene el bloque a int para poder usarlo como parametro de simbolo en invertirBit
        InvertirBit(numero, posicion_a_invertir); // Se aplica la inversión
    }
}
```

Caso 2:

En el caso 2 se aplica encender bit en la posición deseada según el veneno correspondiente. Primero se enciende el bit cero de la máscara, luego se aplica el corrimiento según la posicion deseada y posteriormente se ejecuta un or para que independientemente si estaba pagado el bit se encienda.

De igual forma, El veneno consiste en tomar el bloque inicial de 1000 bytes e ir recorriendo uno por uno de esos bits en la función de encender bit para aplicarle el

debido proceso. Se utiliza la función, atoi para poder introducir el parámetro como entero en encender bit. La siguiente imagen muestra gráficamente como se ejecuta dicho proceso.




```
// Prende el bit deseado segun la posicion dos del byte, y si esta encendido, lo deja igual
void EncenderBit(int& Simbolo, int cual)
{
    int Mascara = 1; // Se pone el bit en 1 que indica que este estara encendido

    Mascara = Mascara << cual; // Se enciende el bit en la posicion cual
    Simbolo = Simbolo | Mascara; // Se realiza un or para poder encender el bit deseado
}

// Se aplica el veneno correspondiente
void Veneno_2(char Bloque[1000], int limite)
{
    int i;

    int posicion_a_encender = 1;

    for (i = 0; i <= limite; i++) {
        int numero = atoi(Bloque);
        EncenderBit(numero, posicion_a_encender);
    }
}
```

Caso 3:

En el caso 3 se aplica apagar bit en la posición deseada según el veneno correspondiente. Primero se enciende el bit cero de la máscara, luego se aplica el corrimiento según la posicion deseada, se niega la máscara con un not y finalmente se ejecuta un and para que independientemente si estaba pagado el bit o prendido, este se apague.

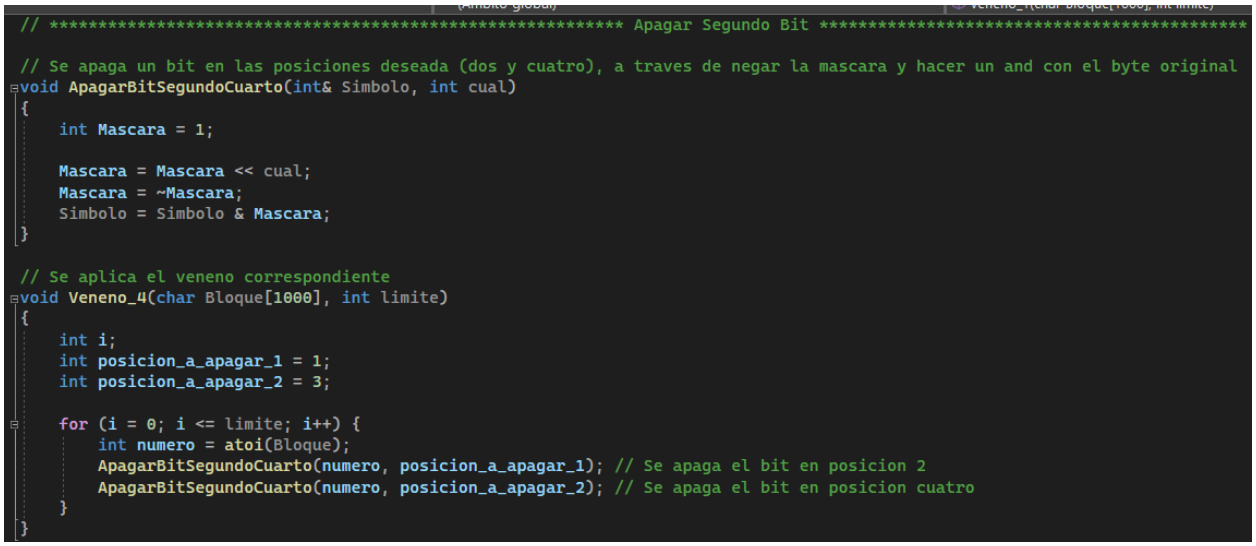
De igual forma, El veneno consiste en tomar el bloque inicial de 1000 bytes e ir recorriendo uno por uno de esos bits en la función de apagar bit para aplicarle el debido proceso. Se utiliza la función, atoi para poder introducir el parámetro como entero en apagar bit. La siguiente imagen muestra gráficamente como se ejecuta dicho proceso.

```
// ********************************************************* Apagar Bit **********************
// Apaga el bit deseado segun la posicion 0 del byte, y si esta apagado, lo deja igual en 0
void ApagarBit(int& Simbolo, int cual)
{
    int Mascara = 1; // Se pone el bit en 1 que indica que este estara encendido

    Mascara = Mascara << cual; // Se enciende el bit en la posicion cual
    Mascara = ~Mascara; // Se niega la máscara
    Simbolo = Simbolo & Mascara; // Se aplica un and con mascara y simbolo(byte original)
}

// Se aplica el veneno correspondiente
void Veneno_3(char Bloque[1000], int limite)
{
    int i;
    int posicion_a_apagar = 0;

    for (i = 0; i <= limite; i++) {
        int numero = atoi(Bloque);
        ApagarBit(numero, posicion_a_apagar);
    }
}
```

Caso 4:

En el caso 4 se aplica apagar bit en la posición deseada según el veneno correspondiente (posición dos y posición cuatro). Primero se enciende el bit cero de la máscara, luego se aplica el corrimiento según la posicion deseada, se niega la máscara con un not y finalmente se ejecuta un and para que independientemente si estaba pagado el bit o prendido, este se apague.

De igual forma, El veneno consiste en tomar el bloque inicial de 1000 bytes e ir recorriendo uno por uno de esos bits. Primero se va a llamar la función para que se apague el bit en la posición dos y posteriormente se aplica en la posición cuatro. Se utiliza la función, atoi para poder introducir el parámetro como entero en apagar bit.



```
// ******************************************************** Apagar Segundo Bit ***************************************

// Se apaga un bit en las posiciones deseada (dos y cuatro), a traves de negar la mascara y hacer un and con el byte original
void ApagarBitSegundoCuarto(int& Simbolo, int cual)
{
    int Mascara = 1;

    Mascara = Mascara << cual;
    Mascara = ~Mascara;
    Simbolo = Simbolo & Mascara;
}

// Se aplica el veneno correspondiente
void Veneno_4(char Bloque[1000], int limite)
{
    int i;
    int posicion_a_apagar_1 = 1;
    int posicion_a_apagar_2 = 3;

    for (i = 0; i <= limite; i++) {
        int numero = atoi(Bloque);
        ApagarBitSegundoCuarto(numero, posicion_a_apagar_1); // Se apaga el bit en posicion 2
        ApagarBitSegundoCuarto(numero, posicion_a_apagar_2); // Se apaga el bit en posicion cuatro
    }
}
```

Caso 5:

El caso 5 constituye una mezcla entre invertir bit y encender bit ya que se invierte la posición cinco del byte y se enciende la posición cuatro. De esta forma se logra crear un encriptado más complejo que los demás.

En la función veneno como se muestra a continuación, se establecen las dos posiciones a ser llamadas en las funciones correspondientes para aplicarle el debido proceso.

```
// ********************************************************* InvertirBit + Encender Bit **********************

// Combinacion de metodo invertir bit y encender bit pero cambia la posicion del bit a psoicion cuatro y cinco
void InvertirBitEncenderBit(int& Simbolo, int cual)
{
    int Mascara = 1;

    Mascara = Mascara << cual;
    Simbolo = Simbolo ^ Mascara;
}

// Se aplica el veneno correspondiente
void Veneno_5(char Bloque[1000], int limite)
{
    int i;
    int posicion_a_invertir_1 = 4; // Se coloca la posicion cinco a invertir 5 dentro del byte
    int posicion_a_invertir_2 = 3; // Se coloca la posicion cuatro a invertir 4 dentro del byte

    for (i = 0; i <= limite; i++) {
        int numero = atoi(Bloque);
        InvertirBitEncenderBit(numero, posicion_a_invertir_1);
        EncenderBit(numero, posicion_a_invertir_2);
    }
}
```

Antídotos

Todos los antídotos funcionan de manera muy similar al veneno ya que constituye un ciclo que tiene como límite la cantidad de leídos del archivo fuente. Recibe la posición o posiciones a invertir en las debidas funciones para revertir el proceso de encriptar.

Caso 1:

En el caso 1 la posición que se utiliza para desencriptar es la posición cero del byte del archivo fuente. La cual utiliza nuevamente la función invertir bit, aplicando el proceso inverso.

```
// El antidoto consiste en aplicarle nuevamente el veneno al archivo generado despues de la inversion, de esta forma se recupera el archivo
void Antidoto_1(char Bloque[1000], int limite)
{
    int i;
    int posicion_a_invertir = 0; // Se coloca la posicion a invertir dentro del byte

    for (i = 0; i <= limite; i++) { // introduce bit a bit
        int numero = atoi(Bloque);
        InvertirBit(numero, posicion_a_invertir); // se restaura el archivo, con el mismo peso
    }
}
```

Caso 2:

En el caso 2 la posición que se utiliza para desencriptar es la posición uno del byte del archivo fuente. La cual utiliza nuevamente la función encender bit, aplicando el proceso inverso.

```
// Se aplica el antidoto correspondiente
void Antidoto_2(char Bloque[1000], int limite)
{
    int i;
    int posicion_a_encender = 1;

    for (i = 0; i <= limite; i++) {
        int numero = atoi(Bloque);
        EncenderBit(numero, posicion_a_encender);
    }
}
```

Caso 3:

En el caso 3 la posición que se utiliza para desencriptar es la posición cero del byte del archivo fuente. La cual utiliza nuevamente la función apagar bit, aplicando el proceso inverso.

```
// Se aplica el antidoto correspondiente
void Antidoto_3(char Bloque[1000], int limite)
{
    int i;
    int posicion_a_apagar = 0;

    for (i = 0; i <= limite; i++) {
        int numero = atoi(Bloque);
        ApagarBit(numero, posicion_a_apagar);
    }
}
```

Caso 4:

En el caso 4 la posición que se utiliza para desencriptar es la posición uno y tres del byte del archivo fuente. La cual utiliza nuevamente la función apagar bit, aplicando el proceso inverso.

```
// Se aplica el antidoto correspondiente
void Antidoto_4(char Bloque[1000], int limite)
{
    int i;
    int posicion_a_apagar_1 = 1;
    int posicion_a_apagar_2 = 3;

    for (i = 0; i <= limite; i++) {
        int numero = atoi(Bloque);
        ApagarBitSegundoCuarto(numero, posicion_a_apagar_1);
        ApagarBitSegundoCuarto(numero, posicion_a_apagar_2);
    }
}
```

Caso 5:

En el caso 5 la posición que se utiliza para desencriptar es la posición cuatro y cinco del byte del archivo fuente. La cual utiliza nuevamente la función invertir bit y encender bit en las posiciones correspondientes, para aplicar el proceso inverso.

```
// Se aplica el antidoto correspondiente
void Antidoto_5(char Bloque[1000], int limite)
{
    int i;
    int posicion_a_invertir_1 = 4;
    int posicion_a_invertir_2 = 3;

    for (i = 0; i <= limite; i++) {
        int numero = atoi(Bloque);
        InvertirBitEncenderBit(numero, posicion_a_invertir_1);
        EncenderBit(numero, posicion_a_invertir_2);
    }
}
```