ANTÍDOTOS Y VENENOS

Manual de Técnico

Descripción breve

A continuación, se realizará un informe detallado sobre la funcionalidad del proyecto Lectura-Escritura-Archivos-C-xBytes.cpp. Además, se indicará como realizar la ejecución del mismo

Estudiantes: Manuel Calero y Brandon Mora

Segundo semestre 2022

Curso: Estructura de Datos

Contenido

Venenos

Los venenos consisten en funciones que utilizan los bits de un archivo y cambian sus valores u orden. Dicho cambio corrompe el archivo, por lo que tratar de abrirlo no es posible sin un antídoto, por eso se llama ***encriptado.*** Para realizar el cambio en los bits simplemente se usan operadores como el AND, OR, XOR y NOT, estos operadores realizan comparaciones con otros valores, por lo que utilizaremos una ***máscara*** para realizar las comparaciones entre ella y cada bit. Dependiendo del operador que sea utilizado, los bits sufren distintas variaciones.

Es primordial explicar la función de los diversos operadores para así tener una mejor comprensión del código. El operador NOT siempre negará los valores, por lo que si a un 1 se le aplica este operador dará como resultado un 0, y si al 0 se le aplica un NOT daría un 1 como resultado. El AND siempre devolverá un 1 siempre que los dos valores comparados sean iguales a 1, en caso contrario se devolverá un 0. En el caso del OR solo dará 0 cuando los dos valores comparados sean iguales a 0, sino devolverá un 1. Por último, tenemos el XOR, este devolverá un 1 siempre que los dos valores comparados sean diferentes, es decir que cuando uno tenga valor 0 y el otro 1, en caso contrario devolverá 0. A continuación se muestra una tabla que resume lo anterior.

AND		
1	1	1
1	0	0
0	1	0
0	0	0

OR		
1	1	1
1	0	1
0	1	1
0	0	0

XOR		
1	1	0
1	0	1
0	1	1
0	0	0

NOT	
1	0
0	1

Funciones por utilizar durante el veneno

Algunas generalidades para todas las funciones es que reciben un char llamado Símbolo (que representará los bits) y un int llamado cual (que representa la posición del bit). Dentro de las funciones se define un entero llamado Mascara inicializado en 1, dicha máscara cumple la función de realizar comparaciones entre él y el Símbolo con los operadores definidos dentro de cada función. La acción que realiza los dos símbolos de ***menor que*** es desplazar la máscara la cantidad de espacios indicados por “cual”.

```
bool BitEncendido(char Simbolo, int cual){
    int Mascara = 1;
    Mascara = Mascara << cual;
    if ((Simbolo & Mascara) == 0)
        return false; // Da 0 lo que signif
    else return true; // Da <>0 lo que sigr
}
```

BitEncendido se encarga de buscar un bit encendido en la posición indicada por “cual” y en caso de encontrar un bit encendido en esa posición se retornará un true, en caso contrario un false. Esto se logra ya que se utiliza el operador OR.

```
void EncenderBit(char& Simbolo, int cual)
{
    int Mascara = 1;
    Mascara = Mascara << cual;
    Simbolo = Simbolo | Mascara;
}
```

EncederBit realiza la acción se encender un bit en la posición indicada por “cual”. Esto se realiza usando el operador NOT y el operador AND.

```
void ApagarBit(char& Simbolo, int cual)
{
    int Mascara = 1;
    Mascara = Mascara << cual;
    Mascara = ~Mascara;
    Simbolo = Simbolo & Mascara;
}
void InvertirBit(char &Simbolo, int cual)
{
    int Mascara = 1;
    Mascara = Mascara << cual;
    Simbolo = Simbolo ^ Mascara;
}
```

invertirBit realiza la acción de invertir el bit recibido. Esto se logra usando el operado XOR. Con invertir bit se refiere a que los 0 ahora valdrán 1 y los 1 valdrán 0.

Funciones secundarias

Además de las funciones anteriores tenemos otras más complejas, que acción que realizan es llamar a más de una de las funciones anteriores. En este caso las funciones solo reciben una referencia al Símbolo, es cual es de tipo char. En estas funciones se hará uso de otro tipo de Mascara, esta estará inicializada para que sea igual a Símbolo. Además se hará llamado de EncenderBit en todas ellas, pasándole Mascara y un valor entero.

```
void InvertirXPosicion(char &Simbolo) {
    int Mascara = Simbolo;
    if (BitEncendido(Simbolo:Mascara, cual:7)) {
        EncenderBit(&:Simbolo, cual:0);
    }
    else {
        ApagarBit(&:Simbolo, cual:0);
    }
    if (BitEncendido(Simbolo:Mascara,cual:0)) {
        EncenderBit(&:Simbolo,cual:7);
    }
    else {
        ApagarBit(&:Simbolo, cual:7);
    }
}
```

InvertirXPosicion realiza la acción de invertir los bits, pero por posición, es decir que el último bit pasa a ser el primer y viceversa, aplicándose así para todos los bits. Para ello se hará un llamado únicamente de las funciones EncenderBit y ApagarBit, para así cambiar el valor de los bits en las posiciones indicadas en cada condición.

```
void correrDerecha(char& Simbolo) {
    int Mascara = Simbolo;//es necesaria usarla como cc
    for (int i = 0; i < 7; i++) {
        if (BitEncendido(Simbolo:Mascara, cual:i + 1)) {
            EncenderBit(&:Simbolo, cual:i);
        }
        else {
            ApagarBit(&:Simbolo, cual:i);
        }
    }
    if (BitEncendido(Simbolo:Mascara, cual:0)) {
        EncenderBit(&:Simbolo, cual:7);
    }
    else {
        ApagarBit(&:Simbolo, cual:7);
    }
}
```

Correr derecha realiza un ciclo for que recorre todos los bits y cambiándole sus respectivos valores. En este caso se cambian todos los valores, pero se guarda el último, ya que si no se realiza este bit se perdería durante los cambios de signos. Dicha función corre los valores hacia un lado, generando así una pérdida del primer elemento, por lo que para evitar esto simplemente se coloca como último,

```
void correrIzquierda(char& Simbolo) {
    int Mascara = Simbolo;//es necesaria usarla como c
    for (int i = 7; i > 0; i--) {
        if (BitEncendido(Simbolo:Mascara, cual:i - 1)) {
            EncenderBit(&:Simbolo, cual:i);
        }
        else {
            ApagarBit(&:Simbolo, cual:i);
        }
    }
    if (BitEncendido(Simbolo:Mascara, cual:7)) {
        EncenderBit(&:Simbolo, cual:0);
    }
    else {
        ApagarBit(&:Simbolo, cual:0);
    }
}
```

Correr izquierda realiza los mismo que la función anterior, cambiando únicamente en donde se comienza el intercambio de los bits. En el caso de la función anterior se comenzaba desde el bit 0, mientras que aquí se comienza en el bit 7. Este de igual forma generaría una perdida de un elemento, por lo que se colocaría en la posición 7, guardando así el elemento original.

Funciones combinadas

```
void correrDerechaUltra(char& Simbolo) {
    for (int i = 0; i < 10; i++) {//corre
        correrDerecha(&:Simbolo);
    }
    InvertirBit(&:Simbolo, cual:5);//invi
    for (int i = 0; i < 7; i++) {//corre
        correrIzquierda(&:Simbolo);
    }
}
void correrIzquierdaUltra(char& Simbolo) {
    for (int i = 0; i < 7; i++) {//corre en
        correrDerecha(&:Simbolo);
    }
    InvertirBit(&:Simbolo, cual:5);//inviert
    for (int i = 0; i < 10; i++) {//corre er
        correrIzquierda(&:Simbolo);
    }
}
```

En este caso se producen dos funciones que llaman tanto a las funciones primarias como secundaras. El correrDerechaUltra se encarga de correr 10 espacios a los bits, luego invierte sus valores con el invertir normal y por último vuelve a correr los elementos, pero

esta vez 7 espacios a la izquierda. En el caso del correrIzquierdaUltra, realiza lo contrario a la función anterior, corriendo así los elementos 7 espacios a la derecha, luego invierte los valores y por último los corre 10 espacios a la izquierda.

Los venenos simplemente llaman a la función que desean, para ello reciben un arreglo de 1000 elementos, equivalente a los bits del archivo a encriptar. También reciben un límite que indica cuando deben para de pasar elementos.

Veneno 1

Este primer veneno únicamente invierte los elementos por su posición.

```
void Veneno_2(char Bloque[1000], int limite)
{
    int i;
    for (i = 0; i <= limite; i++) {
        InvertirXPosicion(& Simbolo:Bloque[i]);
    }
}
```

Veneno 2

Este segundo veneno invierte los bits por su posición y luego invierte el valor de los nuevos bits.

```
void Veneno_3(char Bloque[1000], int limite)
{
    int i;
    for (i = 0; i <= limite; i++) {
        InvertirXPosicion(& Simbolo:Bloque[i]);
        InvertirBit(& Simbolo:Bloque[i], cual:0);
    }
}
```

Veneno 3

El tercer veneno corre los elementos de los bits a la derecha.

```
void Veneno_4(char Bloque[1000], int limite)
{
    int i;
    for (i = 0; i <= limite; i++) {
        correrDerecha(& Simbolo:Bloque[i]);
    }
}
```

Veneno 4

El cuarto veneno corre los bits, pero usando la función correrDerechaUltra.

```
void Veneno_5(char Bloque[1000], int limite)
{
    int i;
    for (i = 0; i <= limite; i++) {
        correrDerechaUltra(& Simbolo: Bloque[i]);
    }
}
```

Antídotos

La función de los antídotos es solucionar lo realizado por los venenos, por lo que simplemente se llamaría la función contraria utilizada en el veneno.

Antídoto 1

Invierte a los bits por su posición.

```
void Antidoto_2(char Bloque[1000], int limite)
{
    int i;
    for (i = 0; i <= limite; i++) {
        InvertirXPosicion(& Simbolo: Bloque[i]);
    }
}
```

Antídoto 2

Primero invierte los bits y luego los vuelve a invertir pero por su posición.

```
void Antidoto_3(char Bloque[1000], int limite)
{
    int i;
    for (i = 0; i <= limite; i++) {
        InvertirBit(& Simbolo: Bloque[i], cual: 0);
        InvertirXPosicion(& Simbolo: Bloque[i]);
    }
}
```

Antídoto 3

Corre los bits a la izquierda.

```
void Antidoto_4(char Bloque[1000], int limite)
{
    int i;
    for (i = 0; i <= limite; i++) {
        correrIzquierda(& Simbolo:Bloque[i]);
    }
}
```

Antídoto 4

Corre los bits a la izquierda, pero usando la función correrIzquierdaUltra

```
void Antidoto_5(char Bloque[1000], int limite)
{
    int i;
    for (i = 0; i <= limite; i++) {
        correrIzquierdaUltra(& Simbolo:Bloque[i]);
    }
}
```

Menú

El menú se encarga de ejecutar los venenos junto a sus debidos antídotos, para ello es necesario que reciba valores, por ello recibe un entero llamado argc cuyo valor es el número de argumentos pasados al programa incluyendo como argumento el nombre del propio programa, además recibe un arreglo de tipo char llamado argv que es quien contiene los argumentos que se han pasado desde el sistema operativo al invocar el programa. Como se van a comprimir archivos es necesario crear dos variables de tipo FILE, la primero que se define se encargaría de abrir una imagen, mientras que la segunda abriría o incluso crearía el destino. Después de ello se abren los archivos y se realizan comprobaciones de si se pueden abrir o no.

```
int main(int argc, char* argv[]) {

    // ...


    for (int i = 0; i < argc; ++i) {
        cout << "argumento " << i << ": " << argv[i] << "\n";
    }
    //---------------------------------------------------------------------

    // Archivos logicos : Buffers tipo FILE
    FILE* ArchivoOrigen, * ArchivoDestino;

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

    /* Apertura del archivo destino, para escritura en binario*/
    fopen_s( _Stream: &ArchivoDestino,  _FileName: argv[3],  _Mode: "wb");
    if (ArchivoDestino == NULL) {
        perror( _ErrorMessage: "No se puede abrir archivo destino"); // , argv[2]
        return -1;
    }
    //---------------------------------------------------------------------
    // PROCESAMIENTO DE ARCHIVOS FUENTE Y DESTINO
    /* Bloque de 1000 bytes, para meter lo que vamos leyendo del archivo */
    char buffer[10000];
```

Se crean dos variables, una de tipo int sin inicializar llamada leídos y una variable de tipo char llamada opción que es igual al argv.

```
int leidos;
char opcion = argv[1][0];
```

La ejecución de los venenos y antídotos consisten en una serie de if y else if, los cules siguen exactamente el mismo proceso, cambiando solo el veneno que llaman. Para que se ejecuten se realizan las comprobaciones de las variables ingresadas en consola y almacenads en opcion, si esta coonicide con alguna de las codiciones entonces se ejecutará el programa. Cada función realiza primer la lectura del archivo origen, luego se ingresa a un ciclo while que no terminará hasta que se haya leído por completo el archivo. Dentro del ciclo ejecutaremos el veneno o antído que deseamos, simplemente de pasaríamos como rederencia al buffer y a leidos. Después de ejecutarse la función anterior, procedemos a guardar el veneno o antído en el archivo destino. Por último cambiamos el valor del leidos por el del archivo origen. Como se mencionó anteriormente, todos realizan la misma acción, así que sólo se explica el primer ejemplo, cuya explicación es útil para las demás condiciones

```
if (opcion == 'e') {
    leidos = fread(_Buffer:buffer, _ElementSize:1, _ElementCount:1000, _Stream:ArchivoOrigen);

    /* Mientras se haya leido algo ... */
    while (leidos > 0)
    {

        Veneno_1(Bloque:buffer, limite:leidos); // Ojo esta usando el mismo metodo de desenci

        /* ... meterlo en el fichero destino */
        fwrite(_Buffer:buffer, _ElementSize:1, _ElementCount:leidos, _Stream:ArchivoDestino);

        /* y leer siguiente bloque */
        leidos = fread(_Buffer:buffer, _ElementSize:1, _ElementCount:1000, _Stream:ArchivoOrigen);
    }
}
else if (opcion == 'd') {
    leidos = fread(_Buffer:buffer, _ElementSize:1, _ElementCount:1000, _Stream:ArchivoOrigen);

    /* Mientras se haya leido algo ... */
    while (leidos > 0)
    {

        Antidoto_1(Bloque:buffer, limite:leidos); // Ojo esta usando el mismo metodo de encri

        /* ... meterlo en el fichero destino */
        fwrite(_Buffer:buffer, _ElementSize:1, _ElementCount:leidos, _Stream:ArchivoDestino);

        /* y leer siguiente bloque */
        leidos = fread(_Buffer:buffer, _ElementSize:1, _ElementCount:1000, _Stream:ArchivoOrigen);
    }
}
```

Una vez realizada la encriptación, procedemos a cerrar los archivos origen y destino.

```
fclose(_Stream:ArchivoOrigen);
fclose(_Stream:ArchivoDestino);
```