



**INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO**



Teoría de Comunicaciones y Señales

“Bajar Volumen a un Archivo WAV”

Abstract

Manipulación en lenguaje C de un archivo multimedia WAV, para bajar el volumen del mismo a la mitad por medio del procesamiento de los datos que componen a la señal digital analizando la cabecera que describe su estructura.

Por:

Romero Gamarra Joel Mauricio

Profesor:

GUTIÉRREZ ALDANA EDUARDO

Agosto 2017

Índice

Contenido

Introducción:.....	1
Análisis Teórico:	1
Software (librerías, paquetes, herramientas):	2
Procedimiento:	3
Resultados	4
Discusión:	9
Conclusiones:.....	10
Referencias:	10
Código	11

Introducción:

Lo que se pretendía al inicio de esta práctica, es bajar el volumen de un archivo wav a la mitad, por lo tanto, había que familiarizarse primero con este tipo de archivos y su manejo en lenguaje C para poder modificar su configuración (específicamente, el volumen).

Para esto se hace un pre-análisis de la cabecera que compone a cualquier archivo de tipo WAV. Un dato interesante de este tipo de archivos es que no utiliza compresión (PCM, uno de los segmentos de su cabecera), esto significa que un archivo pesa mucho más que cualquier otro tipo de archivo del mismo tamaño, además, es un formato que se utiliza profesionalmente ya que no tiene pérdida de calidad si el archivo WAV es grabado a 44,100 Hz y 16 bits por muestra (SampleRate y BitsPerSample de la cabecera respectivamente).²

Análisis Teórico:

Un formato de archivo WAV es un subconjunto de las especificaciones RIFF capaz de almacenar muchos tipos de datos, pero principalmente multimedia (como audio y/o video), está formado por distintos bloques o fragmentos y cada uno de estos está señalado con una etiqueta de 4 caracteres como se muestra a continuación:

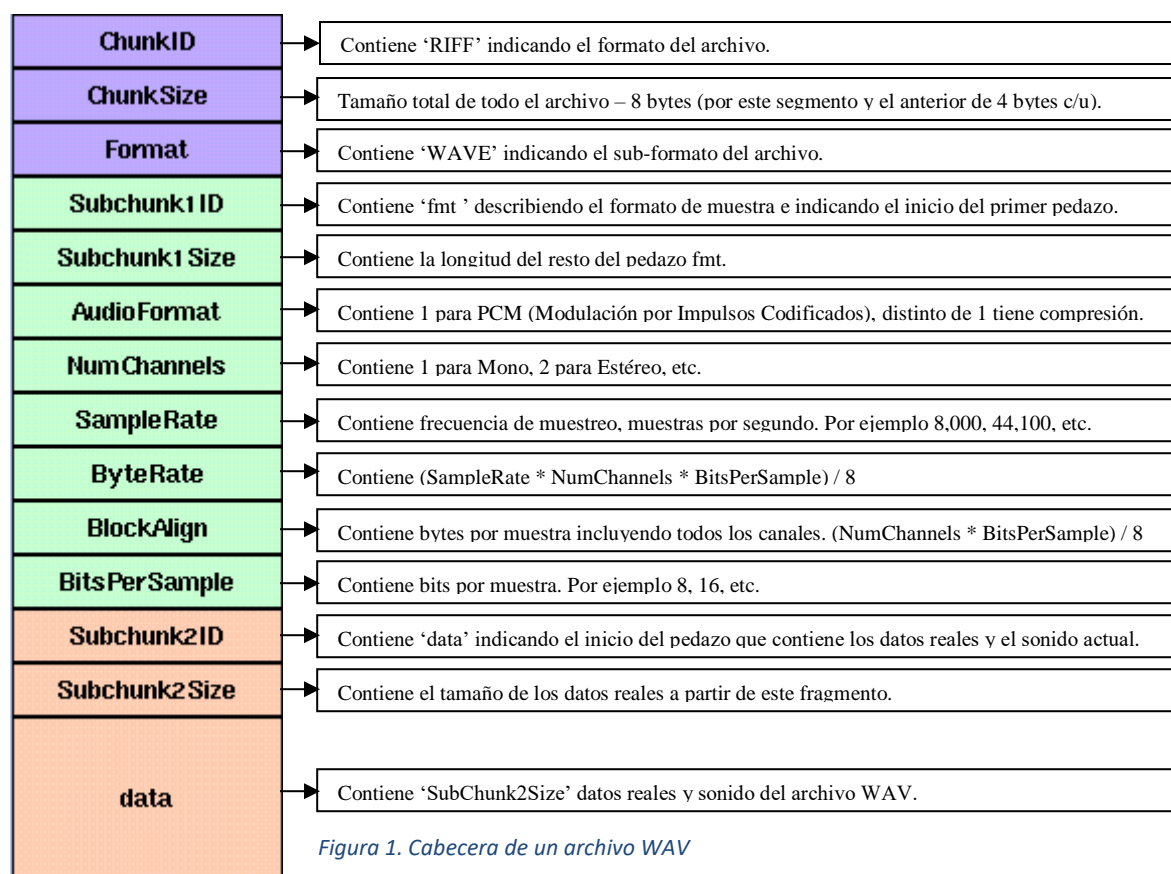


Figura 1. Cabecera de un archivo WAV

El orden que siguen los bytes en un archivo WAV es **Little-endian**, nos podemos dar cuenta con el primer segmento de la cabecera que indica el formato del archivo, ya que, si ese segmento compuesto por 4 caracteres dice “RIFF”, entonces los datos tendrán la distribución **Little-endian**, sin embargo, si ese segmento dice “RIFX”, entonces los datos tendrán la distribución **big-endian**.¹

Como dato, las muestras de 8 bits (en el segmento BitsPerSample), se almacenan en caracteres sin signo (en el caso de lenguaje C, este tipo de dato es **unsigned char**), pero como se especificó en el segmento en la figura 1, puede ser 16 también.

En nuestro caso, los bits por muestra son 16, eso quiere decir que los datos en vez de ir de 0 a 255 (unsigned char), van del -32,767 al 32,767. Esto quiere decir que el tipo de dato en el que guardaremos cada muestra debe tener un tamaño de 2 bytes (16 bits), que en lenguaje C, el tipo de dato a utilizar para guardar cada muestra es **short int**.

El **sonido**, es la **fluctuación de la presión del aire**, cuando se graba un sonido con un micrófono, se cambian las fluctuaciones de la presión del aire por fluctuaciones de voltaje eléctrico (sonido digital), que la tarjeta de sonido de la computadora mide cada cierto tiempo y éstas, se convierten en números, a estos números se le llaman *muestras* y es lo que leemos inmediatamente después del segmento “data” y lo guardamos en enteros cortos (short int) para trabajar con cada una de esas muestras.³

La velocidad con la que la tarjeta de sonido de la computadora mide las fluctuaciones cada cierto tiempo, se le llama **velocidad de muestro**, expresada en Hz (SampleRate de la cabecera, generalmente 44,100, es decir 44100 muestras por segundo para lograr obtener un sonido más fiel).³

Hablando de la cabecera de un archivo RIFF, como se muestra en la Figura 1, está compuesta por muchos trozos, y cada uno de estos, tiene una etiqueta donde empieza (en la figura 1 se observa por colores distintos cada trozo de la cabecera), seguido del tamaño del trozo que la compone, por eso es que tenemos 3, sin embargo el primer tamaño de la cabecera es el tamaño total del archivo (para nuestro caso no nos serviría, a menos que restemos los 44 bytes de la cabecera y ya serían los datos), el segundo tamaño es el resto del trozo ‘fmt’ y el último se refiere al tamaño de los datos (este es el tamaño que nos interesa para realizar la modificación de las muestras, ya que nos indica cuantas muestras tiene el archivo wav).³

Como ya se mencionó anteriormente, un formato de archivo RIFF, almacena los datos como Little – endian. Cada byte está compuesto por 8 bits, y estos a su vez pueden ir ordenados del más significativo al menos significativo o viceversa, en el caso de los Little – endian, los datos son ordenados del menos significativo al más significativo.

En el caso de big – endian, los bits menos significativos se almacenan en la posición más alta de la memoria, es decir van del más significativo al menos significativo. Este tipo de almacenamiento puede ser corregido (de Little – endian a big – endian) utilizando corrimientos como se apreciará en el código.

Software (librerías, paquetes, herramientas):

- Gold Wave versión 4.26⁴
- Frhed versión 1.6.0⁵
- Sublime Text 3⁶
- Bizagi Modeler⁷

Procedimiento:

En la figura 2, se observa el diagrama de flujo general que muestra el funcionamiento del programa para bajar el volumen a un archivo WAV en lenguaje C.

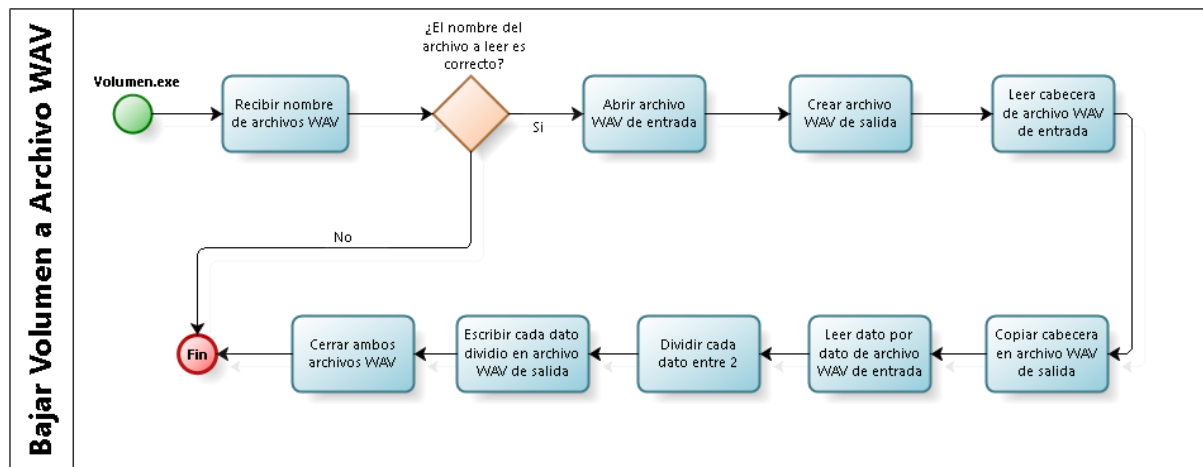


Figura 2. Diagrama de flujo del funcionamiento del programa

Lo primero que debemos hacer es compilar y ejecutar nuestro programa, en este caso, el nombre del archivo de entrada y de salida se reciben al momento de hacer la ejecución, para no utilizar el típico scanf. Por lo tanto, la ejecución del programa suponiendo que llamemos “Volumen” al archivo ejecutable (el archivo .exe), sería de la siguiente forma:

Volumen.exe NombreArchivoEntrada.wav NombreArchivoSalida.wav

Ya que recibimos los nombres de ambos archivos, abrimos ambos en modo binario (el de entrada lo abrimos para lectura y el de salida lo abrimos para escritura, que si ya existe se sobrescribe), posteriormente si el archivo de entrada no se puede abrir o no existe, como en el caso de la Figuras 5 y 6, finalizará el programa y nos imprimirá un mensaje en el que nos dice que ocurrió un error al abrir el archivo como se muestra en la Figura 7.

Ya que los archivos se abrieron correctamente, procedemos a leer la cabecera del archivo de entrada, auxiliándonos de los datos que conocemos de la cabecera de un archivo WAV mostrada en la Figura 1, ya que ya sabemos que tipos de datos debemos de usar únicamente los declaramos dentro de una estructura que los contenga como se muestra en la Figura 3.

Posteriormente, procedemos a copiar la cabecera en el archivo WAV de salida indicado por el usuario, para no afectar nada, y dentro de la misma, recordemos con ayuda del diagrama de la Figura 1, que hay un segmento que nos indica el tamaño de los datos, el cual es nombrado en la Figura 3 como ‘SubChunk2Size’, por lo tanto, el ciclo que haremos para recorrer todos los datos del archivo de entrada es desde 0 hasta ‘SubChunk2Size’, para leer dato a dato (short a short).

```
typedef struct CABECERA
{
    char ChunkID[4];           //Contiene las 'RIFF'
    unsigned int ChunkSize;     //Contiene el tamaño total sin contar este y el segmento anterior (8 bytes)
    unsigned char Format[4];    //Contiene 'WAVE'

    //Aquí comienza el primer subchunk 'fmt'
    char SubChunk1ID[4];       //Contiene 'fmt'
    unsigned int SubChunk1Size; //Contiene el tamaño del resto de el primer subchunk
    unsigned int AudioFormat;   //Formato de audio, es es distinto de 1, es forma de compresión
    unsigned int NumChannels;   //Numero de canales, mono = 1, estereo = 2, etc.
    unsigned int SampleRate;    //8000, 44100, etc.
    unsigned int ByteRate;      //(SampleRate * Numero canales * Bits per Sample) / 8
    unsigned int BlockAlign;    //(Numero canales * Bits per Sample) / 8
    unsigned int BitsPerSample; //8 bits, 16 bits, etc.

    //Aquí comienza el segundo subchunk 'data'
    char SubChunk2ID[4];        //Contiene 'data'
    unsigned int SubChunk2Size; //Numero de bytes en los datos, es decir, bytes despues de este segmento
    unsigned int data;          //Datos de sonidos reales
}cabecera;
```

Figura 3. Estructura utilizada para almacenar cabecera de archivo WAV

Ahora, el ciclo mostrado en la Figura 4 nos muestra la totalidad de la lógica del programa, ya que con un simple `fread`⁸, iremos leyendo dato a dato almacenándolo en la variable 'muestra' de tipo `short`. Inmediatamente después, multiplicamos cada muestra por 0.5 para dividirla a la mitad y posteriormente con un simple `fwrite`⁹ escribiremos 'muestra' de tipo `short` en el archivo wav de salida.

```
for (i = 0; i < (cab.SubChunk2Size); i++)
{
    lectura = fread(&muestra, sizeof (short), 1, archivoEntrada);
    muestra *= 0.5;
    escritura = fwrite(&muestra, sizeof (short), lectura, archivoSalida);
}
```

Figura 4. Ciclo para dividir cada muestra del archivo WAV de entrada a la mitad y escribirlo en el archivo WAV de salida.

Finalmente, cerramos ambos archivos y el archivo WAV de salida habrá quedado dividido a la mitad. En la siguiente sección, se muestran los resultados de la ejecución.

Resultados

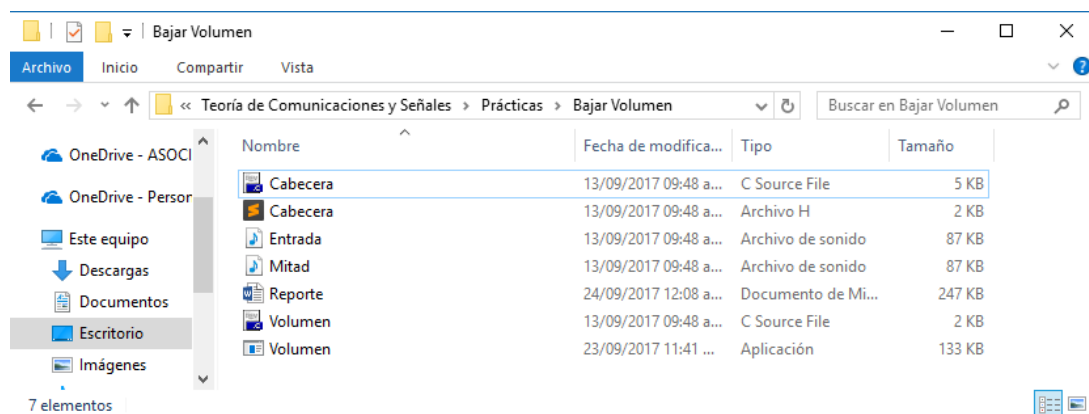
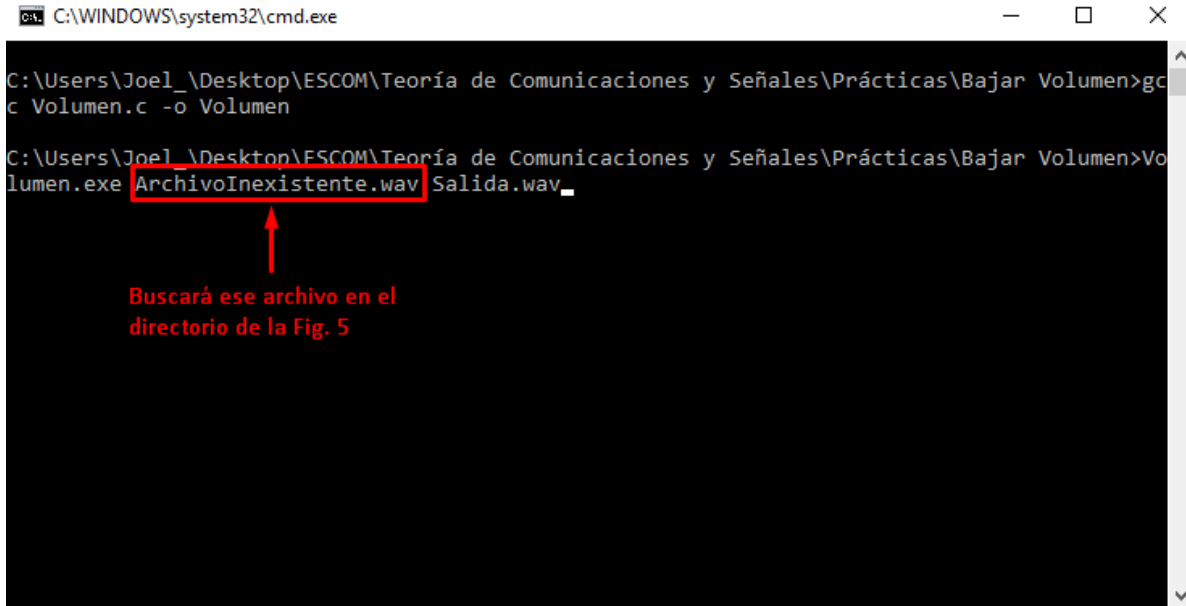


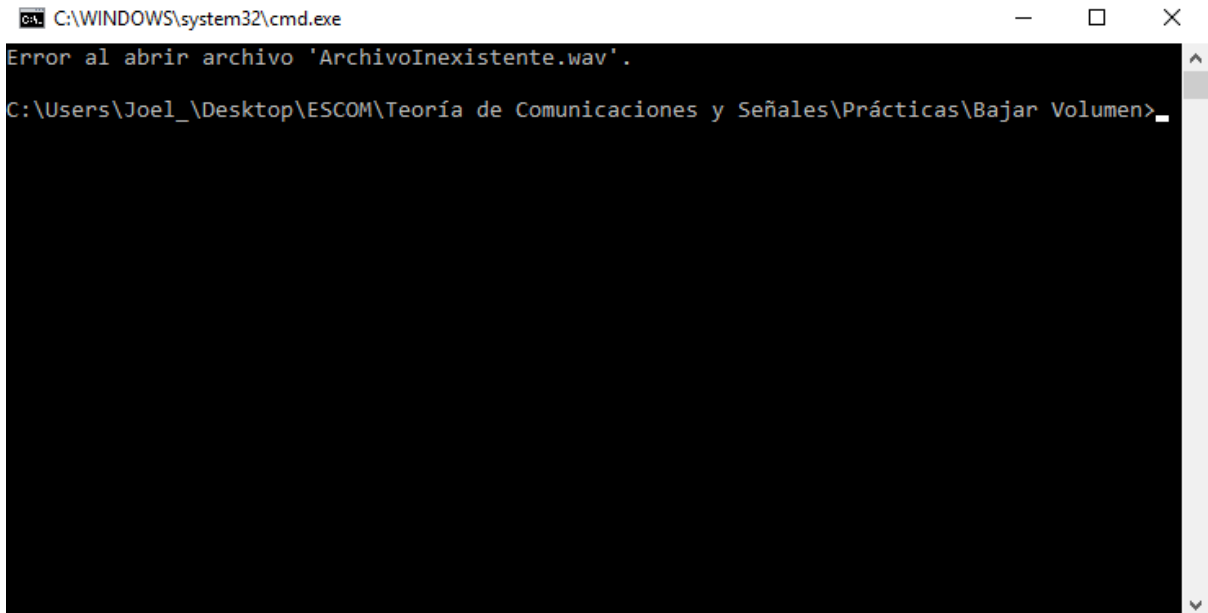
Figura 5. Directorio de donde se obtendrá el archivo WAV a leer y se creará el archivo WAV de salida.



```
C:\WINDOWS\system32\cmd.exe
C:\Users\Joel\Desktop\ESCOM\Teoría de Comunicaciones y Señales\Prácticas\Bajar Volumen>g...
c Volumen.c -o Volumen
C:\Users\Joel\Desktop\ESCOM\Teoría de Comunicaciones y Señales\Prácticas\Bajar Volumen>Vo
lumen.exe ArchivoInexistente.wav Salida.wav_
```

Buscará ese archivo en el directorio de la Fig. 5

Figura 6. Ejemplo de ejecución del programa insertando un archivo que no existe.



```
C:\WINDOWS\system32\cmd.exe
Error al abrir archivo 'ArchivoInexistente.wav'.
C:\Users\Joel\Desktop\ESCOM\Teoría de Comunicaciones y Señales\Prácticas\Bajar Volumen>_
```

Figura 7. Ejemplo de ejecución errónea al no existir el archivo introducido por el usuario en el directorio.

Ahora, mostramos en la Figura 8 el archivo de entrada que leeremos a continuación con GoldWave⁴ nombrado 'Entrada' del directorio mostrado en la Figura 5. Además, en la parte inferior de la Figura 8 podemos ver algunos datos que debe mostrarnos la cabecera de la Figura 10 como: 44,100 de SampleRate, 16 de BitsPerSample, 1 en NumChannels (Mono).

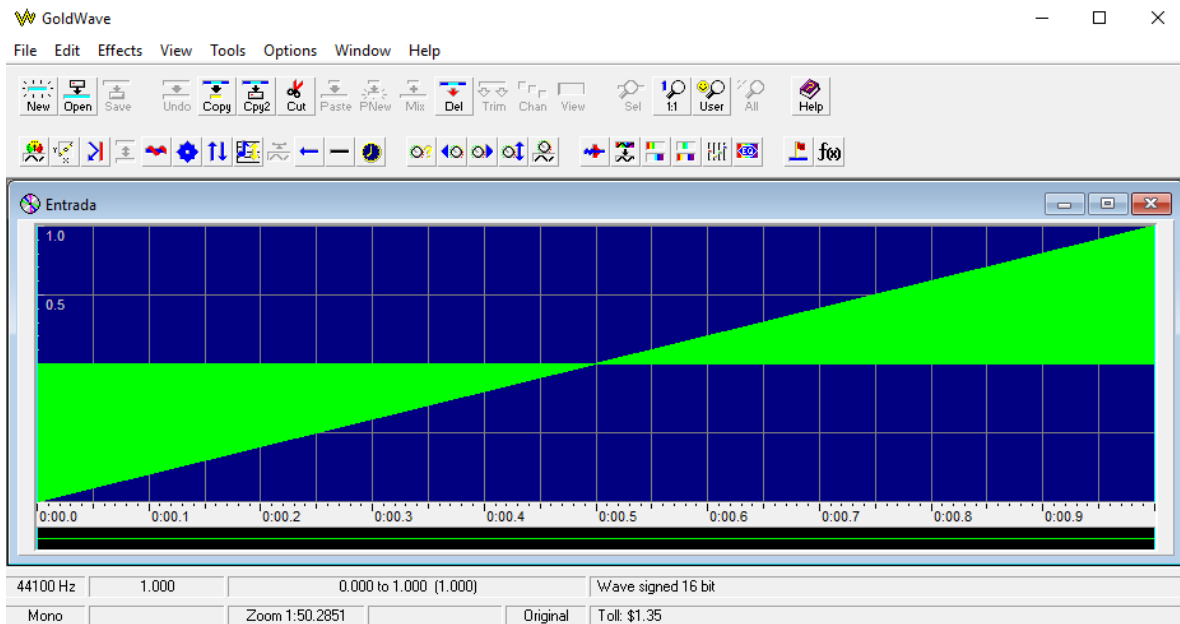


Figura 8. Archivo WAV 'Entrada' visto en GoldWave que se procederá a abrir para bajarle el volumen.

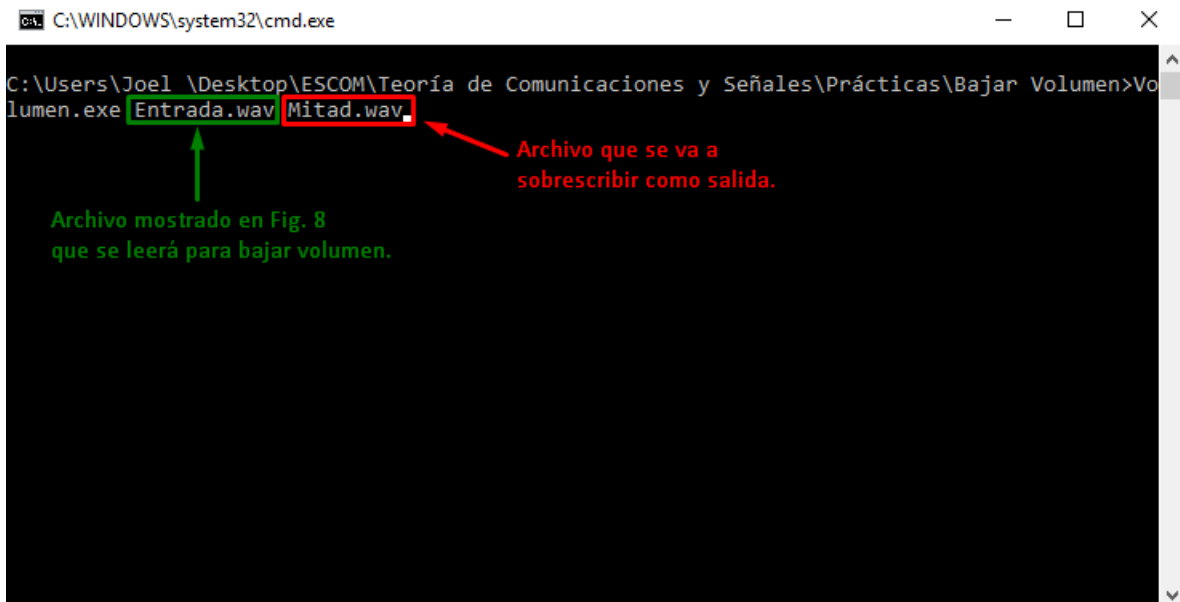


Figura 9. Ejemplo de ejecución correcta del programa con un archivo WAV existente para lectura.

En las figuras 10 y 11, se puede ver como imprimimos los valores que contiene la cabecera del archivo de entrada, para asegurarnos que los datos que obtengamos correspondan a los valores que nos deben de dar obtenidos de la estructura de la cabecera mostrada en la Figura 1.


```
C:\WINDOWS\system32\cmd.exe

(1-4) Chunk ID: RIFF
(5-8) ChunkSize: 88310
(9-12) Format: WAVE
(13-16) SubChunk 1 ID: fmt
(17-20) SubChunk 1 Size: 16
(21-22) Audio Format: 1,PCM
(23-24) Number of Channels: 1, Tipo: Mono → Valor deseado obtenido de GoldWave.
(25-28) Sample Rate: 44100 → Valor deseado obtenido de GoldWave.
(29-32) Byte Rate: 88200 BitRate: 705600
(33-34) Block Align: 2
(35-36) Bits Per Sample: 16 → Valor deseado obtenido de Goldwave.
```

Figura 10. Primera parte de la cabecera del archivo WAV de entrada.

```
C:\WINDOWS\system32\cmd.exe

(23-24) Number of Channels: 1, Tipo: Mono
(25-28) Sample Rate: 44100
(29-32) Byte Rate: 88200 BitRate: 705600
(33-34) Block Align: 2
(35-36) Bits Per Sample: 16
(37-40) SubChunk 2 ID: data
(41-44) SubChunk 2 Size: 88200 ← Numero de muestras en el archivo de entrada.

Archivo 'Entrada.wav' modificado correctamente. ← Todo se realizó correctamente.
C:\Users\Joel_\Desktop\ESCOM\Teoría de Comunicaciones y Señales\Prácticas\Bajar Volumen>
```

Figura 11. Segunda parte de la cabecera del archivo WAV de salida.

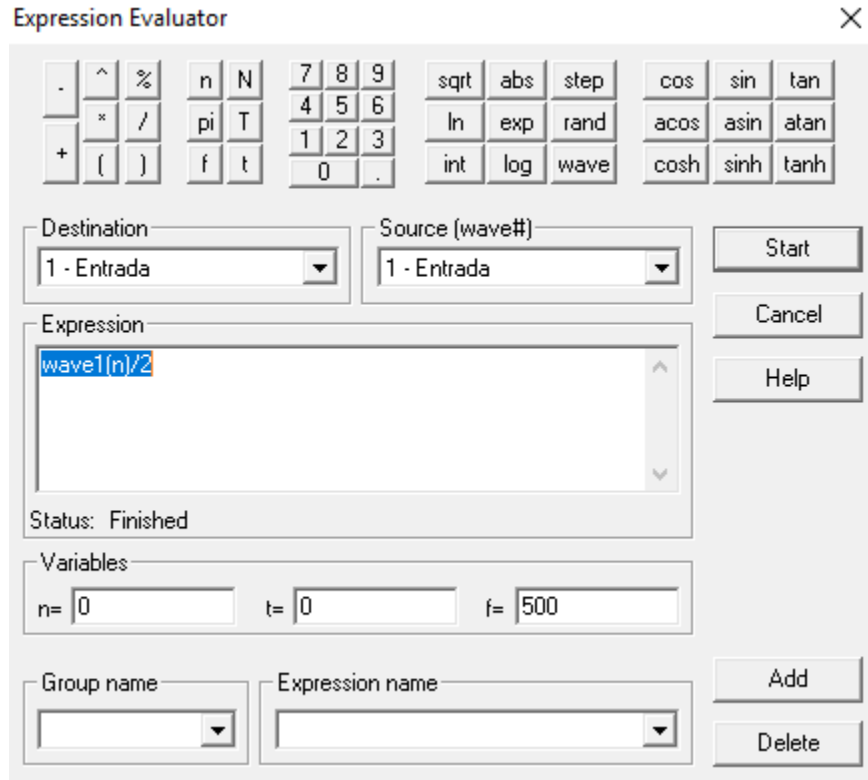


Figura 12. Fórmula por usar para saber la señal que queremos obtener con el programa.

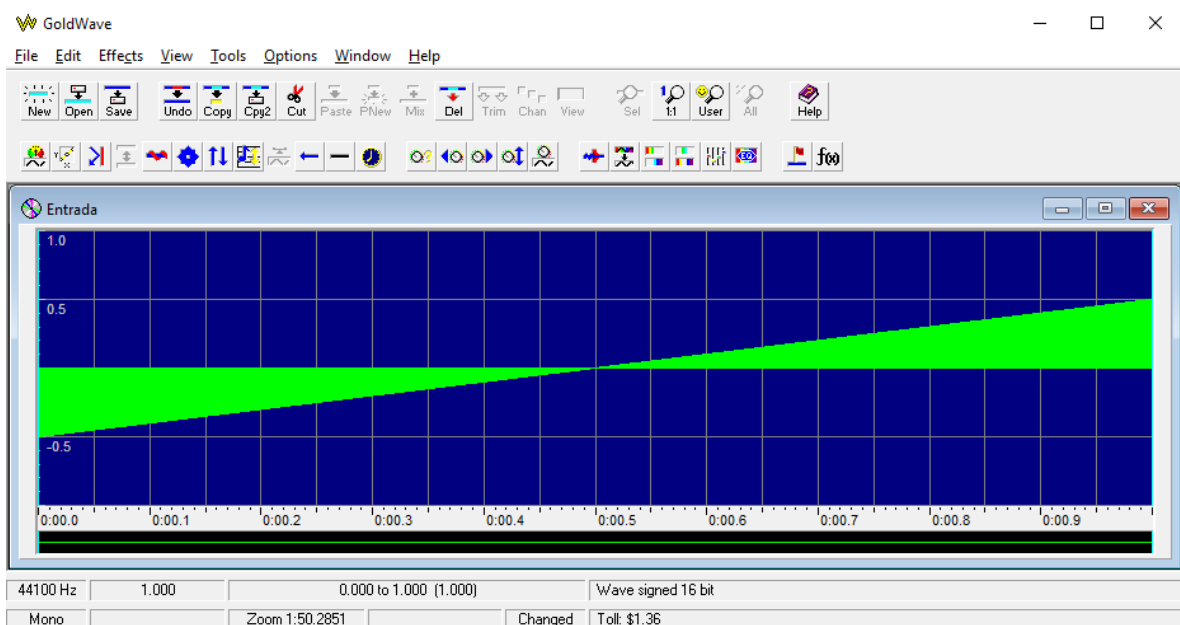


Figura 13. Señal deseada mostrada en GoldWave que debemos obtener con el programa realizado.

En las figuras 12 y 13, mostramos la señal 'Entrada' a la mitad que es lo que debemos obtener con el programa realizado, la señal se muestra en la figura 14.

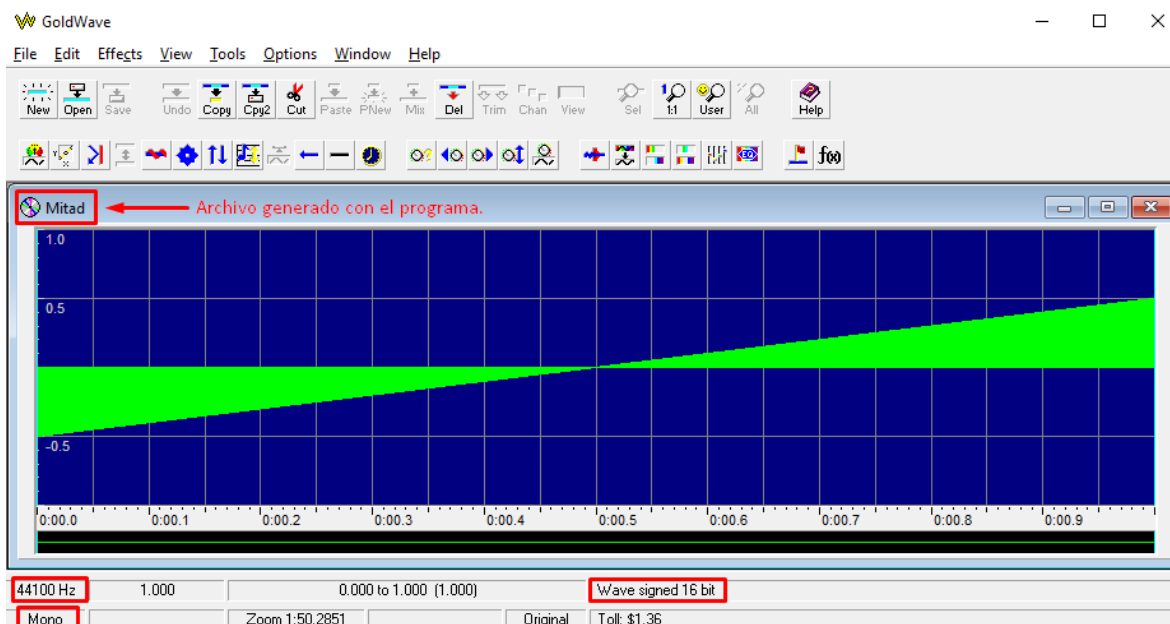


Figura 14. Señal 'Mitad' obtenida con el programa.

En la figura 14 se ve que la señal es prácticamente la deseada (Figura 13), y que los datos mostrados en GoldWave de la cabecera son exactamente los mismos que de la señal original 'Entrada' (Figura 8).

Discusión:

El resultado obtenido con el programa fue el deseado, sin embargo, fue un poco complicado al principio comprender la estructura de la cabecera de un archivo WAV, ya que jamás había leído un archivo de ese tipo en lenguaje C, únicamente archivos de texto.

Uno de los errores que cometí al principio, fue no saber que tipo de dato utilizar para guardar las muestras, ya que comencé utilizando un 'unsigned char', lo cual resultó bastante mal ya que obviamente la señal puede tomar valores negativos, y con ese tipo de dato es imposible, así que fue descartado inmediatamente, después con un char normal pero había un poco de ruido en la señal y no se obtenía el resultado deseado correctamente, solo se podía observar el error al hacer un zoom en la frecuencia y se veían los picos en la señal de salida.

Después de hacer varias pruebas y cambios en el código, comprendí que los datos se almacenaban en un tipo de dato short y después de eso la lógica del programa fue bastante simple ya que únicamente había que dividir cada una de las muestras que se iban leyendo entre 2 para todas las muestras contenidas en el archivo WAV que introdujo el usuario.

Conclusiones:

En lo personal, los archivos WAV son muy interesantes para analizar, ya que como como se mencionó en la introducción, pueden almacenar cualquier tipo de dato multimedia (audio y video), y su manejo en un lenguaje de programación que conocemos (lenguaje C), hace que su manipulación sea bastante fácil una vez que se comprende la cabecera que los compone, como leer los datos y que cosas se puede realizar con ellos.

Por ejemplo, este programa tiene una aplicación en la vida real, ya que ya vimos que un archivo WAV tiene un sonido fiel y profesional, y si tuviéramos un sonido de este tipo podemos manipular el volumen, por ejemplo, un problema común, es al descargar música que suena muy despacio, pues ahora en lugar de bajar el volumen, podemos subirlo y manipular el mismo para obtener un sonido óptimo.

Simplemente, el manejo de este tipo de archivos con una manipulación tan simple en un lenguaje de programación da un potencial enorme, pudiendo (me imagino) lograr hacer cualquier filtro digital que deseemos, modular la señal, en pocas palabras, acondicionarla. Además, teniendo la opción de almacenar un video, imagen, etc. Se me ocurre que podríamos cifrar dato por dato para ocultar un mensaje (Cryptography), que sería tener una comunicación con alguna otra persona por un medio cifrado que proporcionaría seguridad, o por ejemplo procesar una señal para saber por impulsos eléctricos (que al final son altos y bajos en voltaje, y estos pueden ser convertidos en valores digitales) si la persona está agitada (por medio de sensores), o está pensando en algo estresante, etc.

En general, esto nos da un potencial muy grande respecto a lo que se puede lograr, aunque por el momento no conozco mucho del tema ya que es la primera aplicación que realizamos con este tipo de archivos en el curso, sin embargo, me hace imaginarme algunas otras cosas ya mencionadas anteriormente (que probablemente se puedan hacer o no).

En conclusión, este tipo de archivos y su fácil manipulación es interesante, ya que cotidianamente escuchamos música, vemos videos, enviamos fotos, etc. Es decir, tenemos contacto con archivos multimedia, y al tener contacto prácticamente a diario con este tipo de archivos, provoca que su área de aplicación sea muy amplia y se tengan varias formas de procesar las señales y los datos que se ocultan en las mismas para la comunicación.

Referencias:

[1] craig@ccrma.stanford.edu, 'WAVE PCM sounfile format'. [Online]. Disponible en:

<http://soundfile.sapp.org/doc/WaveFormat/>. [Accedido: 09 – agosto – 2017].

[2] Ministerio de Educación, Política Social y Deporte, 'Formatos de audio'. [Online]. Disponible

en: <http://www.ite.educacion.es/formacion/materiales/107/cd/audio/audio0102.html>. [Accedido: 04 – septiembre – 2017].

[3] Timothy John Weber, 'The WAVE File Format'. [Online]. Disponible en:

<http://www.lightlink.com/tjweber/StripWav/WAVE.html>. [Accedido: 09 – agosto – 2017].

[4] Eduardo Aldana Gutiérrez, 'Software' [Online]. Disponible en:

<http://148.204.58.221/ealdana/gwave426.exe>. [Accedido: 09 – agosto – 2017].

[5] SourceForge, 'Frhed', [Online]. Disponible en: <https://sourceforge.net/projects/frhed/files/1.%20Stable%20Releases/1.6.0/>. [Accedido: 09 – agosto – 2017].

[6] Sublime HQ, 'Download', [Online]. Disponible en: <https://www.sublimetext.com/3>.

[7] Bizagi Time to Digital, 'Bizagi Modeler', [Online]. Disponible en: <https://www.bizagi.com/es/productos/bpm-suite/modeler>

Código

Cabecera.h

```
typedef struct CABECERA
{
    char ChunkID[4]; //Contiene las 'RIFF'
    int ChunkSize; //Contiene el
    tamaño total sin contar este y el segmento anterior (8 bytes)
    unsigned char Format[4]; //Contiene 'WAVE'

    //Aquí comienza el primer subchunk 'fmt'
    char SubChunk1ID[4]; //Contiene 'fmt'
    int SubChunk1Size; //Contiene el tamaño del
    resto de el primer subchunk
    short AudioFormat; //Formato de audio, es es
    distinto de 1, es forma de compresión
    short NumChannels; //Numero de canales, mono
    = 1, estereo = 2, etc.
    int SampleRate; //8000, 44100,
    etc.
    int ByteRate; //(SampleRate *
    Numero canales * Bits per Sample) / 8
    short BlockAlign; //(Numero canales * Bits
    per Sample) / 8
    short BitsPerSample; //8 bits, 16 bits, etc.

    //Aquí comienza el segundo subchunk 'data'
    char SubChunk2ID[4]; //Contiene 'data'
    int SubChunk2Size; //Numero de bytes en los
    datos, es decir, bytes después de este segmento
}cabecera;

FILE * abreArchivo (char * nombreArch, char * nombreModificado, int tipo);
//Para abrir los archivos de entrada y salida
void leerCabecera (FILE * archivoEntrada, FILE * archivoSalida, cabecera * cab);
//Para copiar e imprimir la cabecera
```

Cabecera.c

```
#include <stdio.h>
#include <stdlib.h>
#include "Cabecera.h"

FILE * abreArchivo (char * nombreArch, char * nombreModificado, int tipo)
{
    FILE * pt1, * pt2;
    pt1 = fopen (nombreArch,"rb");
    if (pt1 == NULL)
    {
        printf("Error al abrir archivo '%s'.\n", nombreArch);
        exit(0);
    }

    //Abrimos el archivo a escribir en modo binario
    pt2 = fopen (nombreModificado,"wb");
    if (pt2 == NULL)
    {
        printf("Error al crear el archivo '%s'.\n", nombreModificado);
        exit(1);
    }
    if (tipo == 1)
    {
        printf("Archivo '%s' abierto correctamente.\n", nombreArch);
        return pt1;
    }
    else
    {
        printf("Archivo '%s' creado correctamente.\n", nombreModificado);
        return pt2;
    }
}

void leerCabecera (FILE * archivoEntrada, FILE * archivoSalida, cabecera * cab)
{
    //ChunkID
    fread(cab -> ChunkID, sizeof(cab -> ChunkID), 1, archivoEntrada);
    fwrite (cab -> ChunkID, sizeof (cab -> ChunkID), 1, archivoSalida);

    //ChunkSize
    fread(&cab -> ChunkSize, sizeof(cab -> ChunkSize), 1, archivoEntrada);
    fwrite(&cab -> ChunkSize, sizeof(cab -> ChunkSize), 1, archivoSalida);

    //Formato "Fmt"
    fread(cab -> Format, sizeof(cab -> Format), 1, archivoEntrada);
    fwrite (cab -> Format, sizeof (cab -> Format), 1, archivoSalida);

    //SubChunk1ID Formato de datos "fmt"
    fread(cab -> SubChunk1ID, sizeof(cab -> SubChunk1ID), 1, archivoEntrada);
    fwrite (cab -> SubChunk1ID, sizeof (cab -> SubChunk1ID), 1, archivoSalida);

    //SubChunk1Size
    fread(&cab -> SubChunk1Size, sizeof(cab -> SubChunk1Size), 1, archivoEntrada);
    fwrite(&cab -> SubChunk1Size, sizeof(cab -> SubChunk1Size), 1, archivoSalida);

    // Formato de audio
    fread(&cab -> AudioFormat, sizeof(cab -> AudioFormat), 1, archivoEntrada);
    fwrite(&cab -> AudioFormat, sizeof(cab -> AudioFormat), 1, archivoSalida);

    //Canales
    fread(&cab -> NumChannels, sizeof(cab -> NumChannels), 1, archivoEntrada);
    fwrite(&cab -> NumChannels, sizeof(cab -> NumChannels), 1, archivoSalida);

    //SampleRate
    fread(&cab -> SampleRate, sizeof(cab -> SampleRate), 1, archivoEntrada);
    fwrite(&cab -> SampleRate, sizeof(cab -> SampleRate), 1, archivoSalida);
}
```

```

//ByteRate
fread(&cab -> ByteRate, sizeof(cab -> ByteRate),1,archivoEntrada);
fwrite(&cab -> ByteRate, sizeof(cab -> ByteRate),1,archivoSalida);

//Block Align
fread(&cab -> BlockAlign, sizeof(cab -> BlockAlign),1,archivoEntrada);
fwrite(&cab -> BlockAlign, sizeof(cab -> BlockAlign),1,archivoSalida);

//Bits per Sample
fread(&cab -> BitsPerSample, sizeof(cab -> BitsPerSample),1,archivoEntrada);
fwrite(&cab -> BitsPerSample, sizeof(cab -> BitsPerSample),1,archivoSalida);

//SubChunk2ID
fread(cab -> SubChunk2ID, sizeof(cab -> SubChunk2ID),1,archivoEntrada);
fwrite(cab -> SubChunk2ID, sizeof(cab -> SubChunk2ID),1,archivoSalida);

//SubChunk2Size
fread(&cab -> SubChunk2Size, sizeof(cab -> SubChunk2Size),1,archivoEntrada);
fwrite(&cab -> SubChunk2Size, sizeof(cab -> SubChunk2Size),1,archivoSalida);
}

void imprimir_cabecera (cabecera * cab)
{
    char * formatoArchivo = (char *) malloc (sizeof (char));
    printf("\n\n\n(1-4) Chunk ID: %s\n\n", cab -> ChunkID);
    printf("(5-8) ChunkSize: %u\n\n", cab -> ChunkSize);
    printf("(9-12) Format: %s\n\n", cab -> Format);
    printf("(13-16) SubChunk 1 ID: %s\n\n", cab -> SubChunk1ID);
    printf("(17-20) SubChunk 1 Size: %u\n\n", cab -> SubChunk1Size);
    if (cab -> AudioFormat == 1)
        strcpy(formatoArchivo, "PCM");
    printf("(21-22) Audio Format: %u, %s\n\n", cab ->
AudioFormat, formatoArchivo);
    if (cab -> NumChannels == 1)
        strcpy(formatoArchivo, "Mono");
    else
        strcpy(formatoArchivo, "Stereo");
    printf("(23-24) Number of Channels: %u, Tipo: %s\n\n", cab ->
NumChannels, formatoArchivo);
    printf("(25-28) Sample Rate: %u\n\n", cab -> SampleRate);
    printf("(29-32) Byte Rate: %u BitRate: %u\n\n", cab -> ByteRate, cab -> ByteRate*8);
    printf("(33-34) Block Align: %u\n\n", cab -> BlockAlign);
    printf("(35-36) Bits Per Sample: %u\n\n", cab -> BitsPerSample);
    printf("(37-40) SubChunk 2 ID: %s\n\n", cab -> SubChunk2ID);
    printf("(41-44) SubChunk 2 Size: %u\n\n", cab -> SubChunk2Size);
}

```

Volumen.c

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "Cabecera.c"

int main(int argc, char const *argv[])
{
    FILE * archivoEntrada, * archivoSalida;
    cabecera cab;
    int i, lectura, escritura;
    short muestra;
    char * nombreModificado = (char *) malloc (sizeof (char));
    char * nombreArch = (char *) malloc (sizeof (char));
    system ("cls");
    if (argc < 3)

```

```

        printf("Error, faltan argumentos.\n");
    else
    {
        nombreArch = (char *) argv [1];
        nombreModificado = (char *) argv [2];
    }

    //Abrimos los archivos en modo binario
    archivoEntrada = abreArchivo (nombreArch, nombreModificado,1);
    archivoSalida = abreArchivo (nombreArch, nombreModificado, 2);

    //Leemos e imprimimos la cabecera del archivo wav
    leerCabecera (archivoEntrada, archivoSalida, &cab);
    imprimir_cabecera (&cab);

    for (i = 0; i < (cab.SubChunk2Size); i ++)
    {
        lectura = fread(&muestra, sizeof (short), 1, archivoEntrada);
        muestra *= 0.5;

        //Dividimos a la mitad cada uno de los datos
        escritura = fwrite(&muestra, sizeof (short), lectura, archivoSalida);
        //Escribimos los datos nuevos en el archivo
    }
    printf ("\n\n");
    fclose (archivoEntrada);
    fclose (archivoSalida);
    printf ("Archivo '%s' modificado correctamente.\n", nombreArch);
    return 0;
}

```