Para abrir el archivo wav y llegar directamente a los datos del archivo primero tenemos que leer la cabecera y extraer algunos datos de esta para poder hacer algunas operaciones.

**Formato WAVE (RIFF)**

El formato RIFF es un formato Windows para almacenar segmentos (*chunks*) de información multimedia, su descripción, formato, lista de reproducción, etc.

El formato .WAV (*WAVeform Audio File Format*) se almacena dentro de un fichero con formato RIFF, definiendo unos segmentos concretos que puede contener. Por lo tanto es adecuado ver el formato general de un fichero RIFF y qué segmentos se definen para los ficheros .WAV.

**Cabecera RIFF**

El fichero RIFF lleva siempre una cabecera de 8 bytes, que identifica al fichero y especifica la longitud de los datos a partir de la cabecera (esto es, la longitud total menos 8).

Esta cabecera se compone de 4 bytes con el contenido "RIFF" y los otros 4 indican la longitud. Después de la cabecera RIFF siempre hay 4 bytes que identifican el tipo de los datos que contiene; para el caso .WAV estos 4 bytes contienen "WAVE".

struct {

char id[4];

DWORD len;

}riff\_hdr;

char wave\_id[4];

**Chunks RIFF**

Los datos de un fichero RIFF se componen de una secuencia de *chunks*, cada uno de los cuales consta de un campo de identificación de 4 bytes, una longitud de datos de 4 bytes y el conjunto de los datos.

struct {

char id[4];

DWORD len;

} chunk\_hdr;

Al procesar un fichero RIFF, se deben ignorar los segmentos desconocidos, para asegurar compatibilidad con futuras definiciones de los formatos de fichero.

**Generalidades del formato WAVE**

Del conjunto de segmentos que se definen para el formato WAVE existen dos que son obligatorios: el segmento de formato y el segmento de datos. Además el segmento de formato debe aparecer antes que el segmento de datos.

El resto de segmentos son opcionales, aunque como se verá más adelante el segmento FACT es obligatorio en algunos casos.

**Segmento de formato (Format Chunk)**

El segmento de formato identifica por "**fmt**" y se compone de dos elementos: un conjunto de campos comunes y un conjunto de campos específicos.

Este último puede no aparecer, en función del formato concreto elegido. Los campos comunes pueden representarse en la siguiente estructura:

struct {

WORD wFormatTag;

WORD wChannels;

DWORD dwSamplesPerSec;

DWORD dwAvgBytesPerSec;

WORD wBlockAlign;

} CommonFields;

|  |  |
| --- | --- |
| **wFormatTag** | Indica la categoría de formato WAVE del fichero. De este valor depende el contenido del segmento de datos específicos. |
| **wChannels** | Número de canales en el segmento de datos |
| **dwSamplesPerSec** | Tasa de muestreo de cada canal |
| **dwAvgBytesPerSec** | Tasa media de bytes por segundo a la que los datos del fichero WAVE deberían ser transmitidos. Permite estimar el tamaño de buffer a emplear. |
| **wBlockAlign** | Alineación de los bloques del campo datos, en bytes. El software de reproducción necesitará procesar un múltiplo entero de este valor, por lo que permite que el buffer de reproducción esté alineado. |

El software de reproducción debe permitir la existencia de campos específicos y en su caso ignorar todos aquellos campos desconocidos.

### Categorías del formato WAVE

La categoría se especifica a través del campo **wFormatTag**. Los campos específicos del segmento de formato y la representación de los datos en el segmento de datos dependen de este valor.

Actualmente se han definido los siguientes formatos no propietarios:

|  |  |  |
| --- | --- | --- |
| **Formato** | **Valor** | **Descripción** |
| WAVE\_FORMAT\_PCM | 0x0001 | Microsoft Pulse Code Modulation (PCM) |
| FORMAT\_MULAW | 0x0101 | IBM mu-law format |
| IBM\_FORMAT\_ALAW | 0x0102 | IBM a-law format |
| IBM\_FORMAT\_ADPCM | 0x0103 | IBM AVC Adaptative Differential PCM format |

### Formato WAVE\_FORMAT\_PCM

Este formato define un único campo específico que indica el número de bits por muestra de cada canal:

struct {

WORD wBitsPerSample;

} SpecificFields;

Si este campo está comprendido entre 1 y 8 bits, el valor que se almacena en una muestra es un entero sin signo. Si su valor supera los 8 bits entonces el valor almacenado en las muestras es un entero con signo.

Este campo, dividido entre ocho y redondeado al entero mayor más próximo indica el número de bytes por muestra, y permite estimar el campo **wAvgBytesPerSec** como:

wChannels wSamplesPerSecond BytesPerSample

Del mismo modo se puede calcular el valor de **wBloackAlign** de la siguiente forma:

wChannels BytesPerSample

La ordenación de los bytes de datos es la siguiente: los datos se organizan en bloques de **wBlockAlign** bytes. Estos bloques son una secuencia de las muestras para cada canal (en estéreo 0 es izquierdo y 1 es derecho), dentro de cada canal el byte menos significativo va primero.

## Segmento de datos (Data Chunk)

El segmento de datos puede ser de dos tipos: tipo *data* o tipo *data-list*.

El primero es una secuencia de datos sin más elementos. El segundo en una secuencia de dos tipo de segmentos: de datos o de silencio.

Un segmento tipo *data* se identifica por "**data**", mientras que un segmento tipo *data-list* se identifica por "**wavl**". Los segmentos de silencio ("**slnt**") sólo contienen un campo que es una doble palabra con el número de muestras a mantener el silencio.

**NOTA**: El valor de las muestras de silencio no deben ser asignadas a cero, sino al valor de la última muestra reproducida. De lo contrario, puede oírse un 'click' debido al salto que recibe el conversor D/A de salida. Es responsabilidad de la aplicación evitar este 'click' al igual que el que se puede producir al final del silencio.

## Segmento FACT

El segmento FACT tiene como objetivo almacenar información importante sobre los datos del fichero. Es un segmento obligatorio en caso de que el campo de datos sea de tipo *data-list* y con cualquier tipo de compresión.

La definición básica de este segmento contiene un único campo (**dwFileSize**) pero en futuras definiciones del formato WAVE ampliará su contenido por lo que se debe usar el campo longitud de la cabecera del segmento para determinar los campos que están presentes.

## Segmento Cue-Points

Este segmento ("**cue**") tiene como objetivo marcar determinadas posiciones dentro de la forma de onda que contiene el fichero. El segmento consta de un campo **dwCuePoints** indicando el número de puntos a posicionar y una secuencia de estructuras como la siguiente:

struct {

DWORD dwName;

DWORD dwPosition;

FOURCC fccChunk;

DWORD dwChunkStart;

DWORD dwBlockStart;

DWORD dwSampleOffset;

}CuePoints;

|  |  |
| --- | --- |
| **dwName** | Identifica la posición a la que apunta. No debe coincidir con ningún otro elemento de la lista de estructuras. |
| **dwPosition** | Indica la posición de la muestra. Es el número de muestra secuencial dentro del orden de reproducción. [Ver Segmento Playlist] |
| **fccChunk** | Especifica el nombre (o identificación del segmento) que contiene al punto indicado. |
| **dwChunkStart** | Especifica la posición dentro del fichero en la que comienza el segmento que contiene al punto indicado. Es relativo al comienzo del campo de datos del segmento "**wavl**". |
| **dwBlockStart** | Especifica la posición dentro del fichero del comienzo del bloque en que se encuentra la posición apuntada, referenciada con respecto al comienzo del campo de datos del segmento "**wavl**". Su utilidad deriva en que los datos pueden estar comprimidos en bloques. |
| **dwSampleOffset** | Indica el desplazamiento del punto buscado relativo al comienzo del bloque. |

## Segmento Playlist

Especifica un orden de reproducción a partir de diferentes posiciones en el fichero. Se identifica por "**plst**" y contiene un campo **dwSegments**, que indica el número de segmentos que se reproducen y a continuación una lista de estructuras que describen el segmento a reproducir:

struct {

DWORD dwName;

DWORD dwLength;

DWORD dwLoops;

}Playlist;

|  |  |
| --- | --- |
| **dwName** | Especifica el nombre de la posición en la que comienza la sección a reproducir. |
| **dwLength** | Indica el número de muestras a reproducir en esta sección. |
| **dwLoops** | Especifica el número de veces que se reproducirá esta sección. |

## Segmento de datos asociados

Este segmento se identifica por "**adtl**" y permite asociar información a secciones de la forma de onda del fichero. Se define como una lista de otros segmentos que son:

* Etiqueta ("**labl**")
* Nota ("**note**")
* Texto con información de longitud de datos ("**ltxt**")
* Información de fichero embebido ("**file**")

### Segmentos "labl" y "note"

Su formato es similar, "**labl**" contiene una etiqueta o título asociado a una determinada posición de la forma de onda y "**note**" aporta un comentario sobre esa posición.

Los dos segmentos constan de dos campos: un campo **dwName** que identifica el punto de la forma de onda considerado y un campo data que es una cadena terminada con un carácter nula que especifica el contenido de la etiqueta o la nota.

### Segmento "ltxt"

Este segmento contiene información asociada a una porción de datos de una longitud determinada. Los campos de los que se compone son los siguientes: 

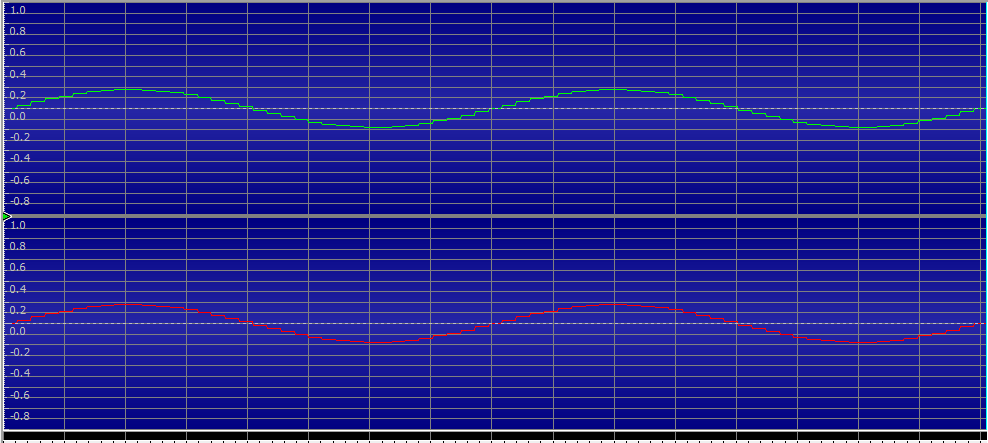
|  |  |
| --- | --- |
| **dwName** | Especifica el nombre de la posición que indica el comienzo de los datos a los que está asociado. |
| **dwSampleLength** | Indica el número de muestras de la porción de interés dentro de los datos sobre la forma de onda. |
| **dwPurpose** | Especifica el propósito del texto. |
| **wCountry** | Especifica el código del país para el texto. |
| **wLanguage** | Especifica el lenguaje empleado. |
| **wCodePage** | Indica el código de página para el texto. |

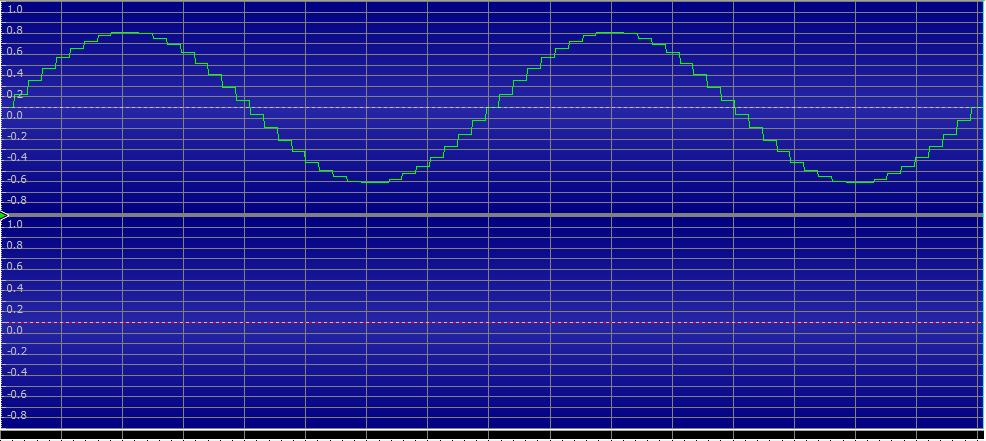
### Segmento "file"

Este segmento contiene información descrita en otros formatos de fichero. Su formato es el siguiente: 

|  |  |
| --- | --- |
| **dwName** | Especifica la posición de la forma de onda a la que se asocia el fichero. |
| **dwMedType** | Especifica el tipo de fichero que se encuentra en el campo **fileData**. Si el fichero es un tipo RIFF, este campo contiene el identificativo del tipo particular de fichero RIFF que sea. |
| **fileData** | El contenido del fichero. |

Para esta práctica el programa abrirá el archive wav y le subirá el volumen y lo reescribirá en otro archive wav, todo implementado en c.





#include <stdio.h>

void Multi(FILE \*s1, FILE \*res);

typedef unsigned short int WORD;

typedef unsigned int DWORD;

typedef unsigned char CHAR;

typedef struct

{

CHAR riff[4];

DWORD tamano;

char wave[8];

DWORD nbytescabeza;

WORD formatopcm;

WORD monoestereo;

DWORD frecnyquist;

DWORD bytesporseg;

WORD bytesporcap;

WORD bitsmuestra;

CHAR data[4];

DWORD ndatos;

} DESCRIPTOR;

main(int argc, char \*\* argv){

FILE \*s1, \*res;

char nom[50];

s1 = fopen(argv[1],"rb");

while(s1==NULL){

printf("Error. Archivo no encontrado!!!\n");

printf("Ingrese el nombre del archivo wav1: ");

scanf("%s",nom);

s1 = fopen(nom,"rb");

}

res = fopen("resultado.wav","wb");

Multi(s1, res);

}

void Multi(FILE \*s1, FILE \*res){

DESCRIPTOR cabS1, cabS2, cabRes;

FILE \*hola;

hola = fopen("hola.txt", "w");

int i,j;

unsigned short int aux, resI;

unsigned int lenS1;

fread (&cabS1, sizeof(cabS1), 1, s1);

cabRes = cabS1;

lenS1 = cabS1.ndatos/2;

printf("Longitud: %d\n",lenS1);

fseek(s1, 44, SEEK\_SET);

fwrite(&cabRes,sizeof(cabRes),1,res);

for(i=0;i<(lenS1/2);i++){

fread(&aux, sizeof(aux),1,s1);

resI = aux\*4;

fread(&aux, sizeof(aux),1,s1);

fwrite(&resI,sizeof(short),1, res);

resI = 0;

fwrite(&resI,sizeof(short),1, res);

}

fclose(hola);

fclose(s1);

fclose(res);

}