



Lab PAV Práctica 1

Procesado de señales por Bloques

**Alejandro Amat
Pablo Díaz
Pol Serra**

INTRODUCCIÓN

En la práctica 1, dividida en dos partes. Se trata de tanto instalar y configurar wavesurfer, como configurar el sistema operativo con las distintas configuraciones que utilizaremos durante el laboratorio. También veremos la distribución de la información en los ficheros .wav. Mientras que en la segunda parte de dicha práctica ya hemos realizado pequeños programas usando Visual Studio Code que calculan algunos estadísticos de la señal de audio para posteriormente mostrarlos gráficamente.

En los siguientes apartados de la memoria detallaremos la resolución con algunas observaciones realizadas de las tareas propuestas en el enunciado de la práctica.

Wavesurfer:

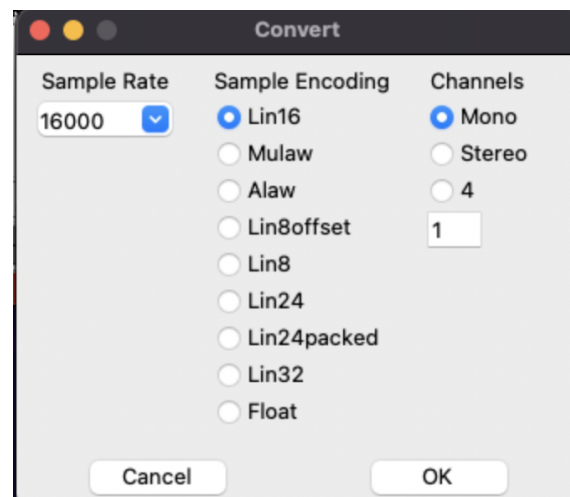
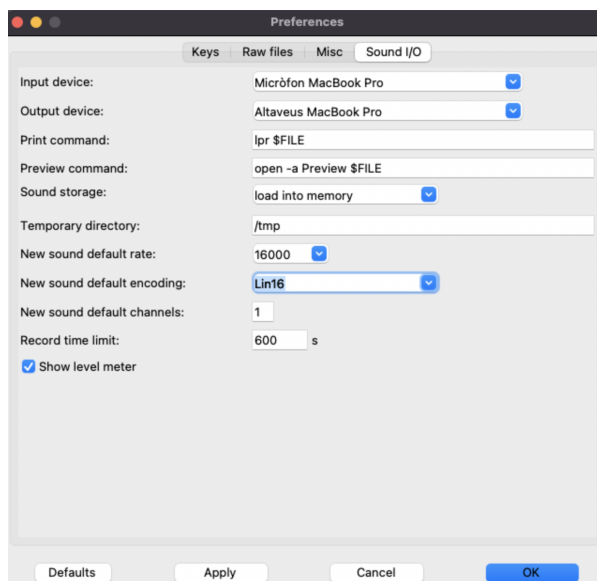
Tarea

Usando **wavesurfer**, grabe un párrafo que empiece por su nombre y el de su compañero de prácticas y, a continuación, una o dos frases cortas con pausas intermedias. Ajuste el nivel de grabación para que la señal tenga un nivel suficiente pero sin saturar. Colóquese adecuadamente el micrófono para que no aparezcan ruidos por roces o golpes.

Guarde la señal en un fichero Microsoft Wave con el formato siguiente: un solo canal (mono), codificado con PCM lineal de 16 bits y una frecuencia de muestreo de 16 kHz. Para ello, puede seguir dos procedimientos distintos:

1. Fijar el formato por defecto en la pestaña **Sound I/O** de la opción **Preferences** del menú **File**. A partir de ese momento, todas las señales que se graben tendrán ese formato y serán guardadas con él.
2. Convertir la señal al formato deseado usando la opción **Convert** del menú **Transform** y, a continuación, guardar el fichero.

Después de descargar e instalar wavesurfer, una vez hecha la grabación escogemos el formato de guardado como de grabación de la señal estipulado por el enunciado.



Visualización/edición de ficheros binarios con Bless

Tarea

Localice los siguientes campos de la cabecera del fichero WAVE grabado anteriormente:

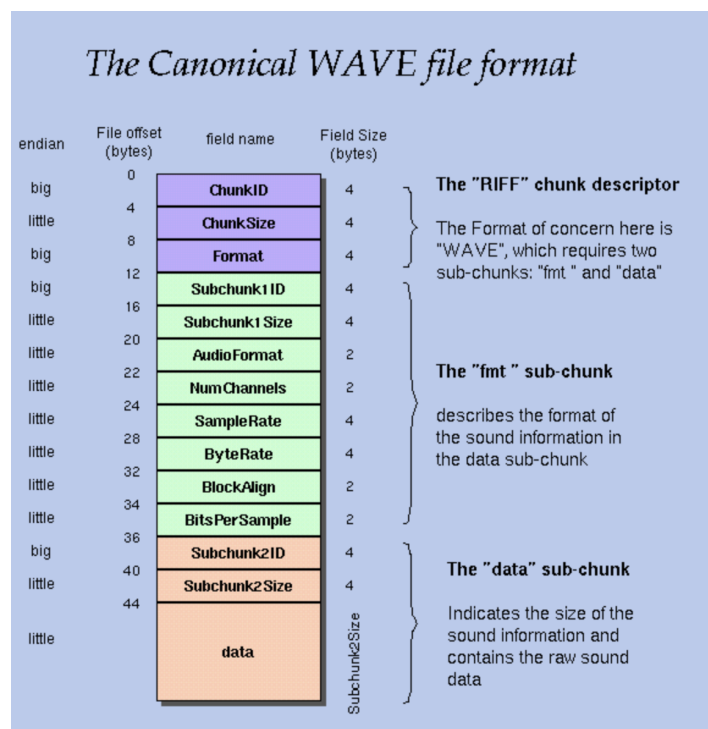
- Descriptores de los chunks y subchunks: "RIFF", "fmt " y "data".
- Número de canales del fichero (será un entero de dos bytes que empieza en el byte 22).
- Frecuencia de muestreo (será un entero de cuatro bytes con offset 24 bytes).

Determine el número de muestras del fichero a partir de la lectura del tamaño en bytes del subchunk "data".

Tal y como está explicado en el enunciado de la práctica, en la columna derecha se muestra el offset del byte de la primera fila, mientras que en la del media los bytes en cuestión y finalmente en la última columna los datos en formato ASCII por lo que evidentemente al no ser un fichero de texto, los datos en formato ASCII no tienen ningún sentido.

Audio2.wav	52	49	46	46	D4	70	04	00	57	41	56	45	66	6D	74	20	10	00	RIFF.p..WAVEfmt ..
00000012	00	00	01	00	01	00	80	3E	00	00	00	7D	00	00	02	00	10	00>...}.....
00000024	64	61	74	61	B0	70	04	00	00	00	00	00	00	00	00	00	00	00	data.p.....
00000036	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Para el análisis del formato .wav podemos ver en la imagen siguiente los distintos campos de información que lleva y su posición en el archivo.



Descriptores de los chunks y sub-chunks “RIFF”, “fmt” y “data”

El RIFF se encuentra en los primeros 12 bytes, y contiene ChunkID, ChunkSize, Format. El sub-chunk fmt contiene los campos en verde de la foto anterior, empieza en el offset 12 y tiene un tamaño 24 bytes. Finalmente, en sub-chunk data con offset 36 bytes, contiene la información del audio además de la información del tamaño. Por lo que el tamaño de este sub-chunk depende de la longitud del propio audio.

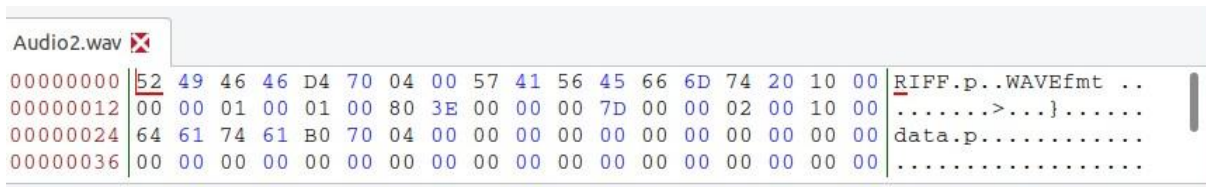
Número de canales, frecuencia de muestreo y tamaño en bytes de muestras:

Sabemos que el número de canales tendrá un tamaño de 2 bytes, un offset de 22 y se encontrará en el bloque NumChannels. La frecuencia de muestreo es de 4 bytes de tamaño, un offset de 24 y su información estará en el bloque SampleRate. Por último, el tamaño en bytes de las muestras, estará en el bloque Subchunk2Size, con un tamaño de 4 bytes y un offset de 40.

Para extraer esta información miraremos el contenido numérico de los bytes de cada campo, mirando el campo “Unsigned”.

Visualización de descriptores “chunks” y “subchunks”:

Descriptor Chunk “ASCII”:



00000000	52	49	46	46	D4	70	04	00	57	41	56	45	66	6D	74	20	10	00	RIFF.p..WAVEfmt ..
00000012	00	00	01	00	01	00	80	3E	00	00	00	7D	00	00	02	00	10	00>...}.....
00000024	64	61	74	61	B0	70	04	00	00	00	00	00	00	00	00	00	00	00	data.p.....
00000036	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Si nos colocamos en la posición offset 0 podemos ver la palabra “RIFF”

Descriptor Subchunk fmt:

Si ahora nos vamos a la posición con offset 12 podemos ver que es correcto.

Descriptor Subchunk “data”:

Podemos ver ahora en la posición 36 el sub-chunk “data”.

Visualización del número de canales del fichero:

```

00 01 00 80 3E 00 00 00 7D 00 00 02 00 10 00 64 61 74 61 B0 70 04 RIFF.p..WAVEfmt .....>...}.....data.p.
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

Sabemos que NumChannels ocupa dos bytes, estando en la posición con offset 22 podemos ver que el valor es “0001”, por tanto 1 canal. Este es el valor esperado ya que nuestra grabación fué guardada con 1 canal.

Visualización de la frecuencia de muestreo:

```

00000000 52 49 46 46 D4 70 04 00 57 41 56 45 66 6D 74 20 10 00 00 00 01 00 01 00 80 3E 00 00 00 7
0000002b 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
00000056 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
00000081 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
000000ac 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
000000d7 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
00000102 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
0000012d 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
00000158 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
00000183 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
000001ae 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
000001d9 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
00000204 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
0000022f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
0000025a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
00000285 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
000002b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
000002db 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
00000306 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
00000331 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
0000035c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
00000387 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
000003b2 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
000003dd 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
00000408 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
00000433 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c
0000045e 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c

```

Signed 8 bit:
Unsigned 8 bit:
Signed 16 bit:
Unsigned 16 bit:

Signed 32 bit:
Unsigned 32 bit:
Float 32 bit:
Float 64 bit:

☒ Show little endian decoding
☐ Show unsigned as hexadecimal

En la posición con offset 24 y sabiendo que la frecuencia de muestreo ocupa 4 bytes, podemos ver el valor “16000” si nos fijamos en en cuadro *unsigned 32 bit*. Esta frecuencia coincide con la que hemos escogido para nuestra grabación.

Visualización del tamaño en bytes de las muestras:

```

B0 70 04 RIFF.p..WAVEfmt .....>...}.....data.p.

```


Tenemos un size en bytes de 290992 (000470B0 en decimal),. Sabiendo que :

$$\text{Subchunk2Size} = \text{NumSamples} * \text{NumChannels} * \text{BitsPerSample}/8$$

Como $n_{\text{canales}}=1$ y usamos 16bits por muestra,
 $n_{\text{muestras}} = \text{Size} * 8 / 16 = \mathbf{145496 \text{ muestras.}}$

Manejo de ficheros de audio con SoX

Tarea:

Estudie las opciones proporcionadas por `sox` y use el programa para convertir el fichero grabado anteriormente al formato de fichero `au` usando codificación ley-mu de 8 bits y una frecuencia de muestreo de 8 kHz. Para ello, ejecute la orden siguiente:

```
sox nombre_fichero.wav -e mu-law -r 8000 nombre_fichero.au
```

Verifique el uso de las opciones de `sox` y compruebe que el fichero resultante cumple con las especificaciones señaladas (por ejemplo, usando el comando `file` de Linux). ¿En cuánto se ha reducido el tamaño del fichero?

Escuche la señal resultante y compruebe la diferencia de calidad con el original.

Hacemos la conversión que proponen y observamos las especificaciones del fichero creado.

```
pablo@pablo-VirtualBox:~/Desktop/PAV$ sox Audio2.wav -e mu-law -r 8000 Audio2.au
pablo@pablo-VirtualBox:~/Desktop/PAV$ file Audio2.au
Audio2.au: Sun/NeXT audio data: 8-bit ISDN mu-law, mono, 8000 Hz
pablo@pablo-VirtualBox:~/Desktop/PAV$
```


Código e Implementación

Tareas:

- Escriba los códigos fuente de `pav_analysis.c`, `pav_analysis.h`, `fic_wave.c`, `fic_wave.h` y `p1.c`.
- Compile y enlace el programa. Corrija los errores de sintaxis y enlazado hasta conseguirlo.
- Modifique `~/.bashrc` (o `~/.profile`, o el que corresponda a su distribución de Linux), para incluir el directorio actual en el path.
- Compruebe que el programa se ejecuta correctamente.

Lo primero, como bien indica el enunciado, era completar los códigos fuente de cara a la correcta funcionalidad del programa.

Pav_analysis.c:

```
1  #include <math.h>
2  #include "pav_analysis.h"
3
4  float compute_power(const float *x, unsigned int N) {
5      float res = 0;
6      for(int i = 0; i < N; i++)
7          res += (x[i] * x[i]);
8
9      return (float) 10*log10(res/N);
10 }
11
12
13 float compute_am(const float *x, unsigned int N) {
14     float res=0;
15     for(int i = 0 ; i < N; i++)
16         res += (float) fabs(x[i]);
17
18     return res/N;
19 }
20
21 float compute_zcr(const float *x, unsigned int N, float fm) {
22     float res=0;
23     for(int i =1; i < N; i++){
24         if((x[i]>0 && x[i-1]<0) || (x[i]<0 && x[i-1]>0) ) //falta considerar caso 0
25             res++;
26     }
27     return res*fm/(2*(N-1));
28 }
29 }
```

Vale la pena mencionar algunos aspectos sobre el código. En cuanto a términos algorítmicos, no hay mucho que comentar puesto que las funciones a implementar pueden resolverse en un par de líneas y siguiendo las indicaciones del PDF no supuso ningún problema. Solo mencionar esta línea de código que no aparece en la captura:

```
if (res==0) res = 0.00000001;
```

Para no hacer logaritmo de 0.

Sin embargo, un aspecto a destacar sería el acceso a las posiciones del vector X . De forma comúnmente usada, utilizamos notación vectorial, sea : $X[i]$.

Sin embargo, aprovechando la funcionalidad que tiene C (así como otros lenguajes) de pasar por referencia los parámetros, existen otras formas de iterar sobre las posiciones del vector.

Hay que entender qué ocurre cuando pasamos un vector como parámetro. En este caso, se indica de forma explícita que lo que estamos pasando es el puntero (o referencia de la dirección de memoria) de la primera posición del vector. Sin embargo lo mismo ocurriría si declaramos el parámetro como un vector.

Teniendo esto en cuenta, en vez de iterar con el formato que hemos usado (el vectorial) podríamos aumentar 4 posiciones de memoria (bytes) en cada iteración, accediendo así a todos los elementos del vector. El formato sería algo así:

```
for (int i = 0; i<N; i++){  
    aux = *( x+4*sizeof(char)*i ) //char es 1 byte en C  
    alternativamente  
    aux = *( x+sizeof(float)*i ) //float, int y long son de 4 bytes en C  
}
```

Pav_analysis.h:

```
p1 > código_p1 > C pav_analysis.h > compute_power(const float *, unsigned int)  
1  #ifndef PAV_ANALYSIS_H  
2  #define PAV_ANALYSIS_H  
3  
4  float compute_power(const float *x, unsigned int N);  
5  float compute_am(const float *x, unsigned int N);  
6  float compute_zcr(const float *x, unsigned int N, float fm);  
7  
8  #endif /* PAV_ANALYSIS_H */  
9
```

fic_wave.c:

```
p1 > código_p1 > C fic_wave.c > abra_wave(const char *, float *, int *, short *, int *)
1  #include <stdio.h>
2  #include "fic_wave.h"
3
4  FILE  *abra_wave(const char *ficWave, float *fm, int *mod, short *channel, int *muestras) {
5      FILE  *fpWave;
6      int fmi; //para leer los 4 bytes (int) de frecuencia
7      short PCM; //leer si es PCM son 2 bytes (short en c) igual que el canal
8      if ((fpWave = fopen(ficWave, "r")) == NULL) return NULL;
9
10     //modulation
11     if(fseek(fpWave, 16 , SEEK_SET)<0) return NULL;
12     fread(mod, 4, 1 , fpWave);
13     fread(&PCM, 2, 1, fpWave);
14     if(*mod!=16 || PCM!=1) return NULL; //SI ES PCM ES 1
15
16     //lectura canal
17     if(fseek(fpWave, 22, SEEK_SET)<0) return NULL;
18     fread(channel, 2, 1 , fpWave);
19     if(*channel != 1) return NULL;
20
21     //lectura frecuencia
22     if (fseek(fpWave, 24, SEEK_SET) < 0) return NULL;
23     fread(&fmi, 4, 1 , fpWave);
24
25     if (fseek(fpWave, 40, SEEK_SET) < 0) return NULL;
26     fread(muestras, 4, 1 , fpWave);
27
28     //posicionamiento en 44 byte para empezar a leer datos
29     if (fseek(fpWave, 44, SEEK_SET) < 0) return NULL;
30     *fm= (float)fmi;
31
32
33     return fpWave;
34 }
35
36 size_t  lee_wave(void *x, size_t size, size_t nmemb, FILE *fpWave) {
37     return fread(x, size, nmemb, fpWave);
38 }
39
40 void  cierra_wave(FILE *fpWave) {
41     fclose(fpWave);
42 }
43
```

Para poder realizar y complementar la función `abra_wave()` , debemos entender qué parámetros contiene y qué acciones debe realizar. Comentar que en esta captura, se muestra también código de ampliación.

Básicamente, se nos pasa el nombre del fichero y una referencia a la dirección de memoria de la variable donde se aloja la frecuencia. Tras el ejercicio de ampliación, existirán 3 parámetros (También punteros) para almacenar el modo, canal y número de muestras.

La funcionalidad se divide en dos: abrir el archivo pasado y crear un puntero a la estructura de datos de C para gestionar ficheros (FILE). En este caso es READONLY, por lo que, bajo esta sintaxis se refiere como “r”. Vale la pena mencionar que a nivel de sistema operativo la información relativa a ficheros abiertos se conoce como descriptor de fichero, que está conectada directamente con la tabla de canales. Aunque no vaya estrictamente de la mano con la tarea es interesante comentar que tanto los directorios como ficheros se estructuran en formato árbol con la concepción de Inodo. Teniendo como prioridad el recuento de enlaces y referencias entre estos (Bajo el nombre de Hard Links y Soft Links).

Por ello es necesario conectar la salida/entrada de datos con uno de estos Inodos (En este caso el fichero de audio). Para ello se relacionan 3 estructuras de datos cruciales en todo Sistema Operativo: Tabla de Inodos (la cargada en memoria/caché), Tabla de Ficheros Abiertos y Tabla de canales.

Por defecto, para cada proceso (No vale la pena ponerse a definir proceso en esta práctica) el canal estándar de lectura es el 0, el de escritura 1 y el de error 2. Aquí es donde entra la función “open”, cuyo flujo es el siguiente. Asocia un inodo a una entrada en la TFA(tabla de ficheros abiertos) donde se almacena el puntero a la información de cada fichero (Que se modifica cuando se escribe/lee o con **lseek**). A su vez, esta entrada en la TFA se relaciona con una en la tabla de canales (una de escritura y una de lectura, si es el caso). Es por eso, que el sistema operativo al hacer open, es capaz de relacionar su sistema I/O con un fichero (inodo) concreto.

La otra funcionalidad, que viene muy de la mano con lo recién explicado, es el poder leer del fichero pasado por parámetro para extraer la información de frecuencia, canal, modo y muestras. Para ello se tiene en cuenta la estructura de cabecera mencionada anteriormente.

Con este fin, se utiliza la función lseek() para modificar los valores del puntero de la TFA() y read para leer por el canal relacionado (Que en este caso será el 3 dado que el proceso no abre ningún otro canal)

La estructura es:

```
if(fseek(fpWave, 16 , SEEK_SET)<0) return NULL;
    fread(mod, 4, 1 , fpWave);
    fread(&PCM, 2, 1, fpWave);
    if(*mod!=16 || PCM!=1) return NULL; //SI ES PCM ES 1
```

En este caso, se accede al byte 16 mediante la función y se extrae la información correspondiente.(Un short y un int)

Cabe mencionar que hemos utilizado lseek tantas veces como íbamos a leer de cara a una mejor comprensión del código, pero es realmente redundante.

Solo haría falta el primero al byte 16 y luego al 40 para las muestras. Ya que del 16 al 24 la lectura se hace consecutiva y del 40 al 44 también.

fic_wave.h:

```
p1 > código_p1 > C fic_wave.h > abre_wave(const char *, float *, int *, short *, int *)
1  #ifndef FIC_WAVE_H
2  #define FIC_WAVE_H
3
4  FILE  *abre_wave(const char *ficWave, float *fm, int *mod, short *channel, int *muestras);
5  size_t lee_wave(void *x, size_t size, size_t nmemb, FILE *fpWave);
6  void  cierra_wave(FILE *fpWave);
7
8  #endif /* FIC_WAV_H */
9
```

Tarea:

Edite el fichero Makefile y compruebe el funcionamiento de make. En concreto, compruebe que si alguno de los códigos fuentes de p1 o de los ficheros de cabecera cambia, se ejecutan las órdenes justas y necesarias para reconstruir el programa; y, si ninguno de ellos cambia, make simplemente no hace nada.

Antes de compilar el código decidimos hacer el makefile para tener más comodidad a la hora de hacer pruebas:

```

p1 : p1.o pav_analysis.o fic_wave.o
    gcc -o p1 p1.o pav_analysis.o fic_wave.o -lm

p1.o: p1.c pav_analysis.h fic_wave.h
    gcc -c p1.c

fic_wave.o: fic_wave.c fic_wave.h
    gcc -c fic_wave.c

pav_analysis.o: pav_analysis.c pav_analysis.h
    gcc -c pav_analysis.c

```

Por ello una vez ya con todo, probamos el programa (En una versión inicial):

```

alejandro@alejandro-ZenBook-Pro-15-UX550GD-UX550GD:~/Desktop/CFIS/4A/PAV/practic
as/PAV/p1/código_p1$ make
gcc -c fic_wave.c
gcc -o p1 p1.o pav_analysis.o fic_wave.o -lm

```

No se modifican todos los archivos porque no es la primera vez que ejecuto el make en la captura.

p1 Audio2.wav

896	-59.821934	0.000883	452.830200
897	-62.581978	0.000606	855.345886
898	-60.118851	0.000832	704.402527
899	-62.714775	0.000615	955.974854
900	-62.951363	0.000598	855.345886
901	-65.618057	0.000437	1207.547119
902	-64.456863	0.000475	1157.232666
903	-63.091202	0.000600	704.402527
904	-66.029778	0.000399	1459.119507
905	-58.889290	0.000920	704.402527
906	-63.375851	0.000546	1006.289307
907	-62.641571	0.000611	855.345886
908	-65.306717	0.000426	1006.289307

FM = 16000.00 Hz, mod= 16 PCM, channel= 1 (mono), bytes = 290992

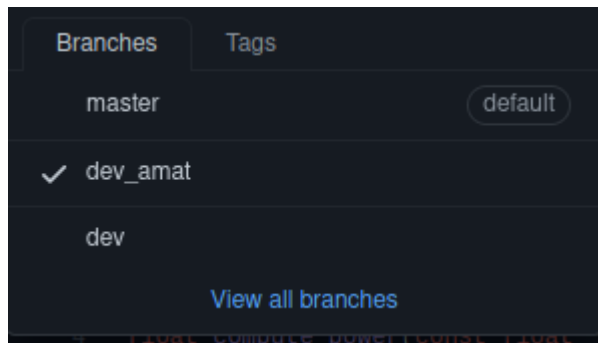
Tareas:

- Inicialice, de modo global, la información del usuario de Git.
- Inicialice la gestión de versiones para el proyecto de la primera práctica.
- Escriba el fichero `.gitignore` de manera que Git sólo gestione el código fuente de PAV/P1.
- Realice la primera confirmación (*commit*) con los ficheros originales de la práctica.

Ya teníamos git configurado en nuestro ordenador por lo que la inicialización no la podemos documentar. Sin embargo, comentar que tras crear el repositorio, decidimos crear tres ramas : Master, dev y release (que posteriormente modificamos y llamamos dev_amat ya que solo la usaría Alejandro para hacer pequeñas modificaciones). Siguiendo con el patrón de desarrollo, todo debe hacerse en desarrollo y debe realizarse una PR, donde al menos uno del equipo debe aceptar.

```
● alejandro@alejandro-ZenBook-Pro-15-UX550GD-UX550GD:~/Desktop/CFIS/4A/PAV/practicas/PAV$ git branch
  dev
* dev_amat
  master
○ alejandro@alejandro-ZenBook-Pro-15-UX550GD-UX550GD:~/Desktop/CFIS/4A/PAV/practicas/PAV$
```

Estas son las ramas de mi local, que son las mismas que las del repositorio:



Para definir la restricción de PR obligatoria a master y la imposibilidad de hacer un merge directo, se definen una serie de reglas en la configuración de seguridad.

Protect matching branches

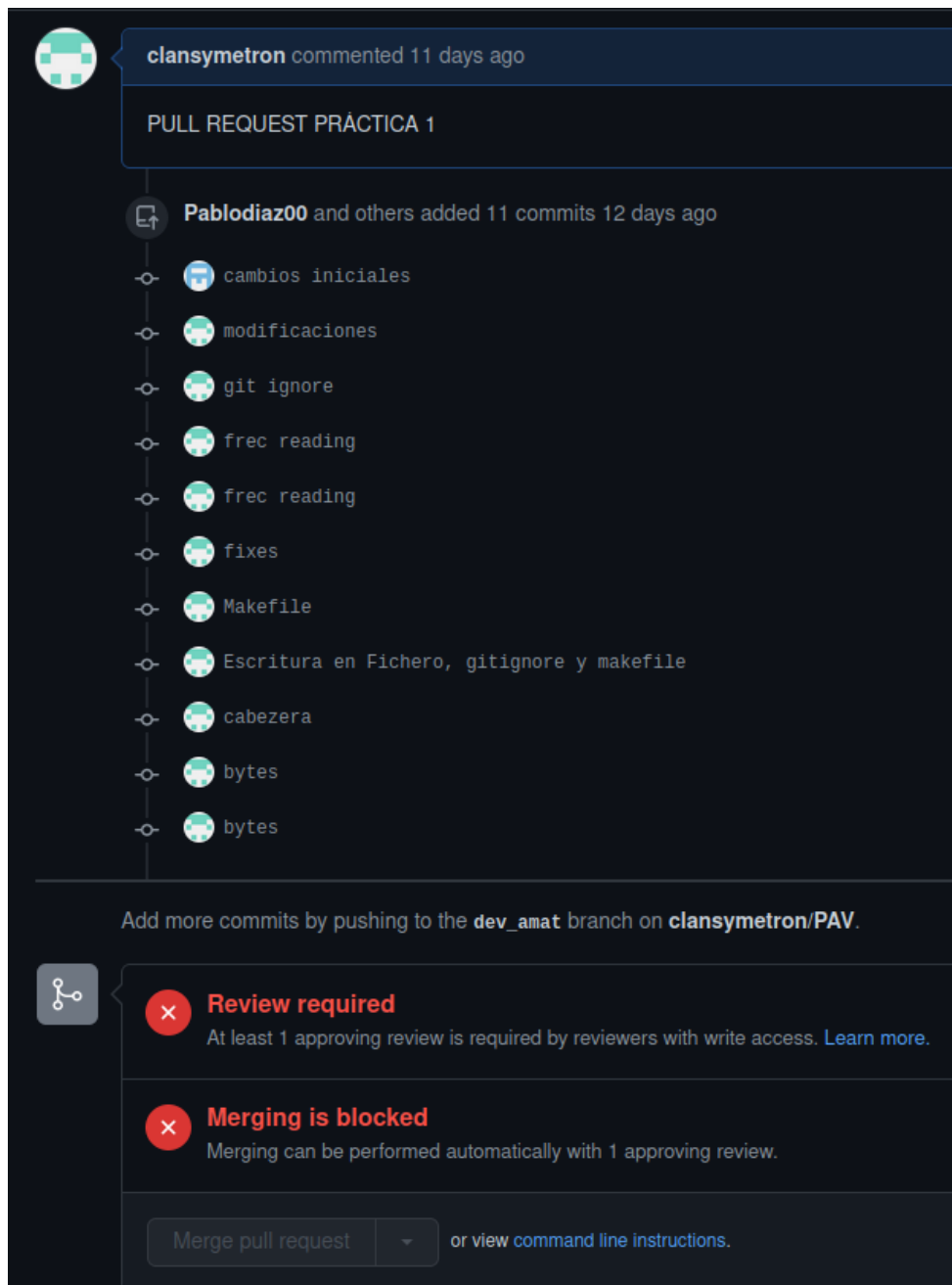
- ☒ **Require a pull request before merging**
When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.
- ☒ **Require approvals**
When enabled, pull requests targeting a matching branch require a number of approvals and no changes requested before they can be merged.
Required number of approvals before merging: 1 ▾
- ☐ **Dismiss stale pull request approvals when new commits are pushed**
New reviewable commits pushed to a matching branch will dismiss pull request review approvals.
- ☐ **Require review from Code Owners**
Require an approved review in pull requests including files with a designated code owner.

- ☐ **Require status checks to pass before merging**

- ☒ **Do not allow bypassing the above settings**
The above settings will apply to administrators and custom roles with the "bypass branch protections" permission.

No tenemos la captura del primer commit ya que no sabíamos que se debía hacer memoria, pero adjuntamos captura de un commit de otro momento:

```
alejandro@alejandros-ZenBook-Pro-15-UX550GD-UX550GD:~/Desktop/CFIS/4A/PAV/practicas/PAV$ git add .
alejandro@alejandros-ZenBook-Pro-15-UX550GD-UX550GD:~/Desktop/CFIS/4A/PAV/practicas/PAV$ git commit -m "bytes"
[dev amat dfa4656] bytes
2 files changed, 1 insertion(+), 1 deletion(-)
alejandro@alejandros-ZenBook-Pro-15-UX550GD-UX550GD:~/Desktop/CFIS/4A/PAV/practicas/PAV$ git push
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 12 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 902 bytes | 902.00 KiB/s, done.
Total 6 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To https://github.com/clansymetron/PAV.git
  930b59a..dfa4656  dev_amat -> dev_amat
```



Vemos en esta imagen el historial de commits realizados antes de hacer la Pull Request. De esta forma, como se observa, aún no se ha aceptado ni 'mergeado' con Master.

Por último, comentar el archivo gitignore:

```
p1
fic_wave.o
p1.o
pav_analysis.o
respuesta.txt
```

2. Modifique p1.c para que el programa acepte un argumento, que será el fichero de audio de entrada, o dos, en cuyo caso el segundo será un fichero de texto en el que se escribirá el resultado del análisis.

Adjuntamos solo los trozos de código modificados o que valgan la pena resaltar:

```
int main(int argc, char *argv[]) {  
    float durTrm = 0.010;  
    float fm;  
    int N;  
    int trm;  
    float *x;  
    short *buffer;  
    int mod;  
    short channel;  
    int muestras;  
    FILE *fpWave;  
    FILE *fpResult;  
    FILE *stdout_P1 = stdout;
```

Se agregan algunas variables para la gestión de escritura en ficheros y lectura de datos de cabecera.

```
//lectura 2 parametro.  
if(argc ==3){  
    fpResult = fopen(argv[2], "w");  
    stdout_P1 = fpResult; //Hacemos que la salida estándar sea el descriptor de fichero de escritura  
}
```

Creamos las estructuras de datos correspondientes a la gestión de otro fichero y asignamos a la 'salida estándar' del proceso el descriptor de escritura del fichero.

```

while (lee_wave(buffer, sizeof(*buffer), N, fpWave) == N) {
    for (int n = 0; n < N; n++) x[n] = buffer[n] / (float) (1 << 15);

    //Al no ser binario se puede hacer print sobre el descriptor de fichero
    fprintf(stdout_P1, "%d\t%f\t%f\t%f\t%f\n", trm, compute_power(x, N),
        compute_am(x, N),
        compute_zcr(x, N, fm));
    trm += 1;
}
fprintf(stdout_P1, "FM = %.2f Hz, ", fm);
fprintf(stdout_P1, "mod= %d PCM, ", mod);
fprintf(stdout_P1, "channel= %d (mono), ", channel);
fprintf(stdout_P1, "bytes = %d ", muestras);
cierra_wave(fpWave);
free(buffer);
free(x);

```

El punto 1 y 3 ya se han comentado en el apartado de código.

4. Use su programa p1 para obtener los parámetros estadísticos de la señal grabada en el laboratorio y visualícelos con `wavesurfer`.
 - Compare el resultado obtenido para la potencia media con el calculado por `wavesurfer`. ¿A qué se deben las diferencias?

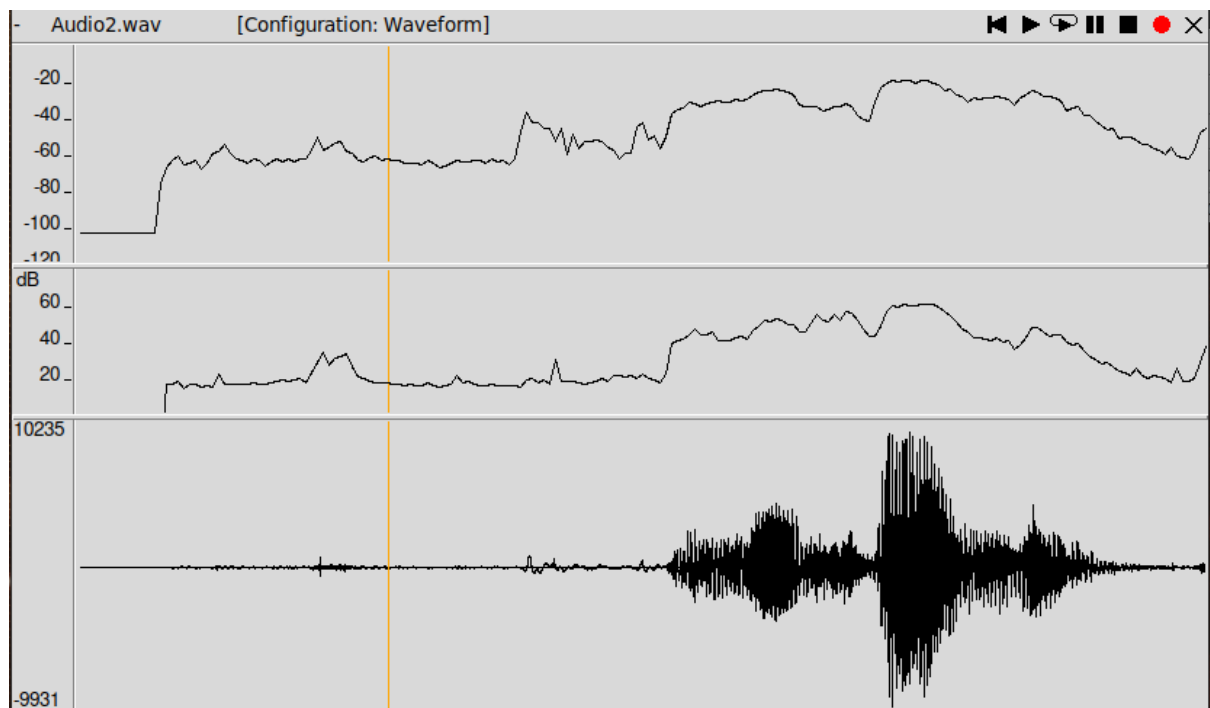
Es posible que deba cambiar los parámetros de `wavesurfer` para poder realizar esta comparación.

```
as/PAV/p1/código_p1$ padsp wavesurfer
```

```
as/PAV/p1/código_p1$ p1 Audio2.wav | cut -f2 -> data.pot
```

Nos quedamos únicamente con la potencia. Ahora sí, comparamos con la señal en Wavesurfer. Cabe mencionar, como se ha comentado que cuando un bloque de análisis es todo 0, a la hora de calcular el logaritmo se calcula una potencia en db de -100 (No sabíamos qué criterio aplicar).

Para analizar, observamos la señal en formato Waveform y creamos dos planos. Uno para data plot donde cargaremos nuestro fichero de datos y otro para mostrar la potencia calculada por Wavesurfer.

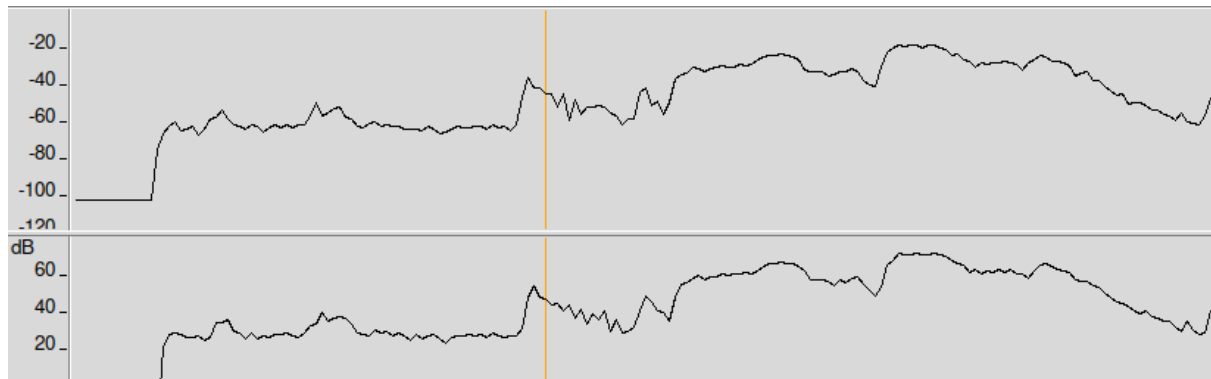


La de arriba es la potencia extraída de nuestros datos. Como observamos tiene una forma muy parecida a la que calcula wavesurfer. Las diferencias observadas son por dos motivos. El primero es ya que nosotros normalizamos nuestra señal ($1 < \leq 15$) con lo que adopta valores entre 1 y -1 (Con lo que los valores están desplazados hacia el negativo) y el otro factor es la ventana de análisis. Dependiendo de esta longitud, podemos tener un valor u otro. (Nuestra N es de 160)

Pane	Data Plot	Power plot	Sound	Playback
Analysis window length:		160	points	
Analysis window type:		Rectangle		
Pre-emphasis factor:		0		
Frame interval:		0.01	s	
Show channel:		all	(0,1,2,...,left,right,all)	
Max power value:		80	dB	
Min power value:		0	dB	
<input type="checkbox"/> Add header in export file				

OK
Cancel
Apply

Aplicando una N de 160 y poniendo a 0 el factor de énfasis tenemos una señal mucho más parecida:



De la ampliación ya hemos mencionado el primer ejercicio (de leer más datos de la cabecera→ muestras, modo y canal) y no hemos logrado hacer el 2 en condiciones.