PRÁCTICA 5

MULTIPLICACIÓN

Teoría de comunicaciones y señales

Profesor: Gutiérrez Aldana Eduardo

Alumno:

Amador Nava Miguel Ángel

Grupo: 3CM8

Índice

Introducción

Se implementará la multiplicación de dos archivos de audios con la misma frecuencia de muestreo, con la misma o distinta duración, donde las posibles combinaciones de multiplicación de los archivos son: mono por mono, mono por estéreo, estéreo por estéreo.

Objetivos

Generales: Implementación de la Multiplicación

Específicos:

- Ver su utilidad en la frecuencia.

Software y equipo utilizado

- Ubuntu 17.10
- Wine 1.7
- GoldWave 4.26
- Sublime Text 3

Marco teórico

Multiplicación de números complejos

Los números complejos $\mathbb{C}$ son una extensión de los números reales ($\mathbb{R}$), los cuales tiene una parte real y una parte imaginaria

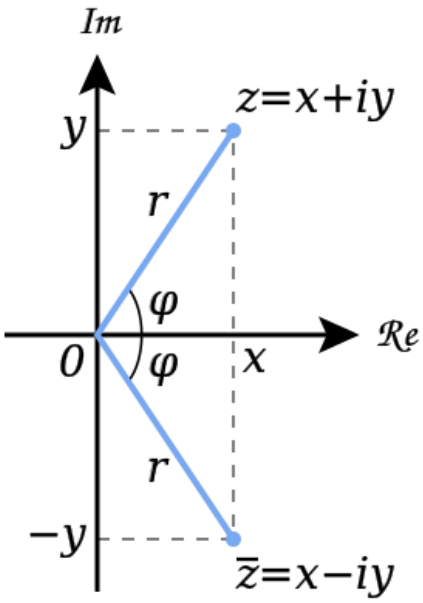

Ilustración 1 Plano complejo

En el conjunto de los complejos se definen tres operaciones:

Suma:

$$(x + iy) + (u + iv) = (x + u) + i(y + v)$$ *Ecuación 1*

Producto escalar:

$$a(x + iy) = (ax + iay)$$ *Ecuación 2*

Multiplicación:

$$(x + iy) * (u + iv) = (x * u - y * v) + i(x * v + u * y)$$ *Ecuación 3*

Preparación del entorno para el desarrollo de la práctica

GoldWave 4.26 fue creado para Windows por lo cual para poder ejecutarlos en Ubuntu 17.10 necesitaremos instalar wine.

Introduciremos los siguientes comandos:

sudo add-apt-repository ppa:ubuntu-wine/ppa

sudo apt-get update

sudo apt-get install wine1.7 winetricks

Una vez instalado wine para ejecutar GoldWave lo haremos de la siguiente manera:

WINEPREFIX~=/.wine32 wine "nombre del ejecutable"

Planteamiento del problema

Se pide realizar la multiplicación de dos archivos donde el canal 1 será la parte real y el canal 2 la parte imaginaria.

Combinaciones:

1. Mono por Mono
2. Mono por Estéreo
3. Estéreo por Estéreo

Análisis Teórico

Cabecera de un archivo wav

Bytes	Campo	Descripción
4	ChunkID	Contiene las letras “RIFF” en ASCII
4	ChunkSize	36 bytes más SubChunk2Size. Este es el tamaño entero del archivo en bytes menos 8
4	Format	Contiene las letras “WAVE”
4	Subchunk1ID	Contiene las letras “fmt”
4	Subchunk1Size	Este es el tamaño del resto del Subchunk
2	AudioFormat	Los valores distintos de 1 indican alguna forma de compresión.
2	NumChannels	Numero d canales: 1= mono, 2=Stereo
4	SampleRate	Número de muestras por segundo
4	ByteRate	SampleRate * NumChannels * BitsPerSample/8
2	BlockAlign	NumChannels * BitsPerSample/8. El número de bytes para una muestra incluyen todos los canales
2	BitsPerSample	Bits por muesta. 8,16,etc.
4	Subchunk2ID	Contiene las letras “data”
4	Subchunk2Size	NumSamples * NumChannels * BitsPerSample/8. Este es el número de bytes en la data
*	Data	Datos del sonido

Tabla 1 Información de la cabecera de un archivo wav

Multiplicación de números complejos

Usaremos la ecuación 3 la cual quedará de la siguiente manera:

Canal 1: $(x * u - y * v)$

Canal 2: $i(x * v + u * y)$

Es importante mencionar que estamos trabajando con datos de tipo short donde 32767 (el valor máximo de un short) teóricamente hablando es 1, y 32768 es -1. Por lo cual tenemos los siguientes casos:

1. Al multiplicar un archivo mono por un mono o mono por estéreo lo máximos valores que tendríamos son:

$$32767*32767=32767^2$$

Pero esto es igual a multiplicar

1*1=1

Por lo tanto después de multiplicar en cualquier operación x*u; y*v; x*v y u*y dividiremos entre 32767.

También se puede presentar el caso de tener:

$$32767*-32768$$

Que es lo mismo que:

1*-1=-1

Por lo tanto se cumple lo mismo, al dividir entre 32767

2. Al multiplicar un archivo estéreo por otro estéreo se presenta un caso crítico cuando tenemos un 1 en ambos archivos en la parte real y en la parte imaginaria, la parte real se cancela, y queda cero pero la parte imaginaria es igual a 2:

Canal 1: $(x * u - y * v) = 1 * 1 - 1 * 1 = 0$

Canal 2: $i(x * v + u * y) = 1 * 1 + 1 * 1 = 2$

Tenemos dos opciones la primera es dividir entre dos la parte real e imaginaria para evitar la pérdida de información o perder la información y cualquier número que sea mayor a 1 o menor a -1 igualarlo a 1 y a -1 respectivamente.

Resultados

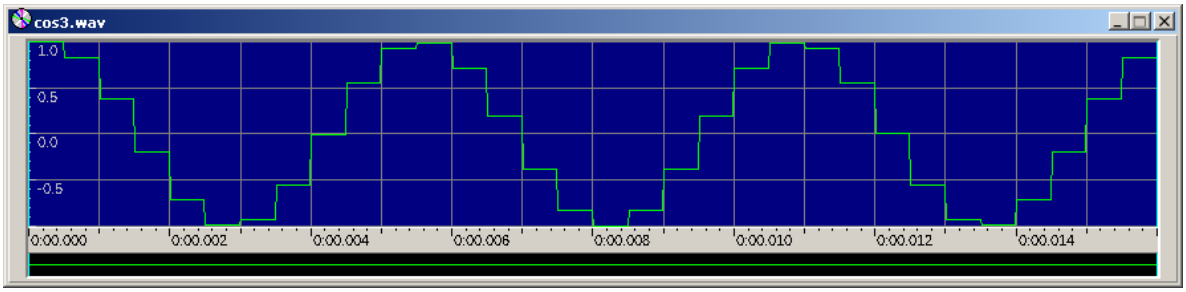

Ilustración 2 Coseno de 187.5 Hz

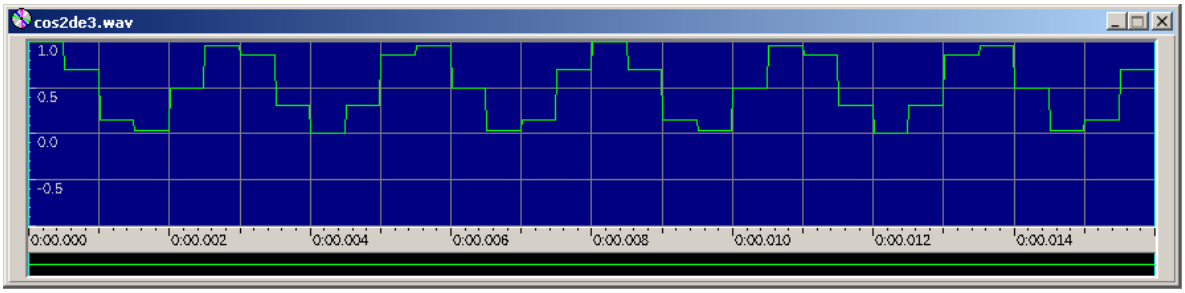

Ilustración 3 Resultado de multiplicar el coseno de 187.5 Hz por sí mismo

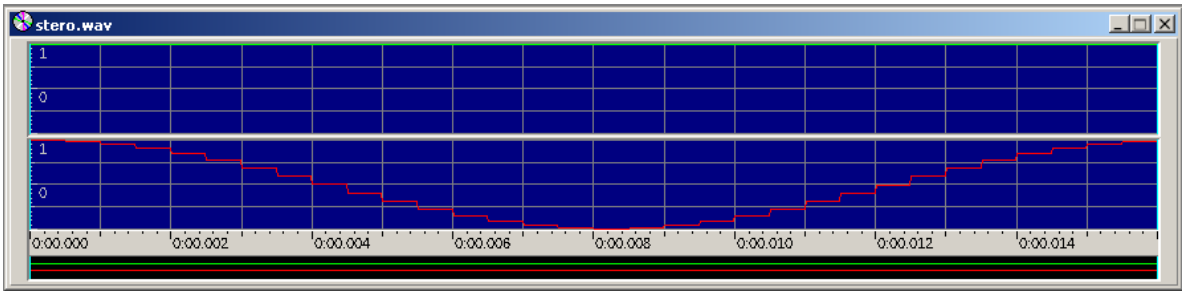

Ilustración 4 Archivo estéreo donde el canal 1 es igual a 1 y el canal 2 es el coseno de 62.5 Hz

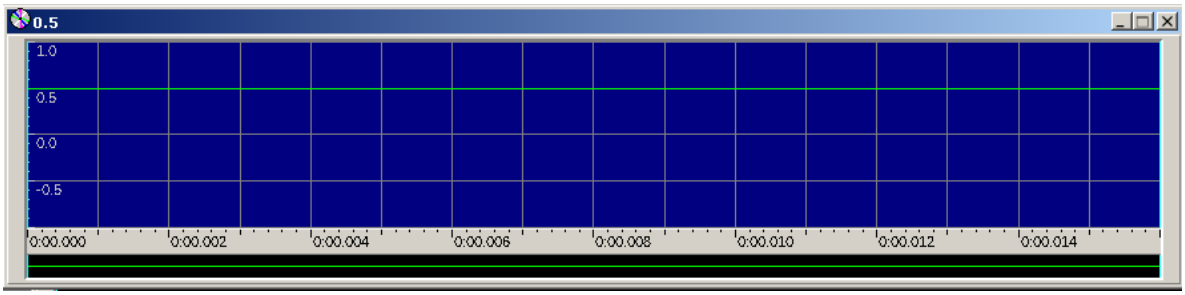

Ilustración 5 Archivo mono con valor de 0.5

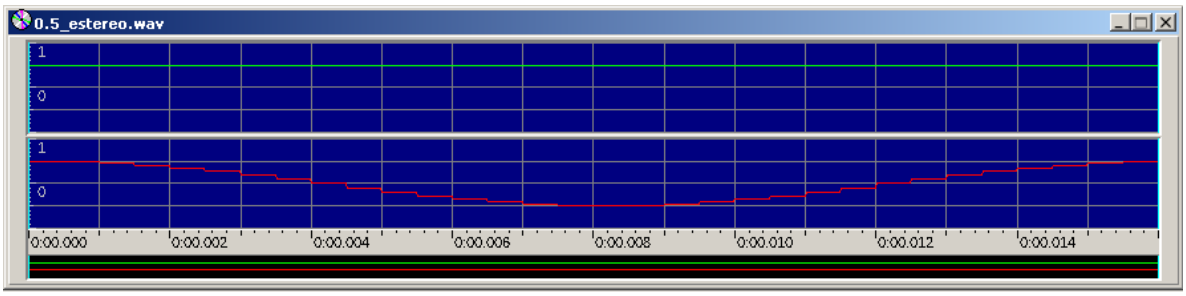

Ilustración 6 Resultado de multiplicar el archivo de la ilustración 4 y 5

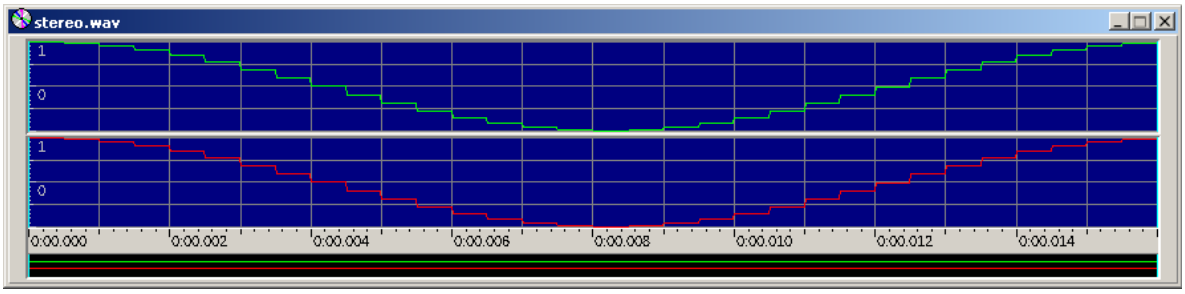

Ilustración 7 Coseno de 62.5 Hz en canal 1 y2

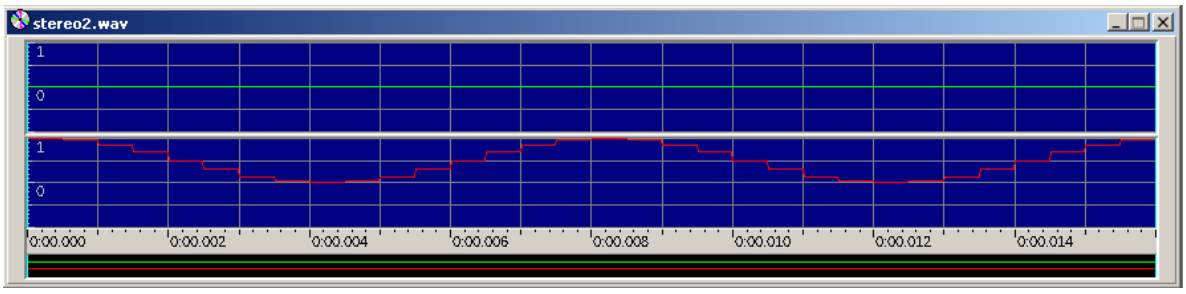

Ilustración 8 Resultado de elevar al cuadrado la señal de la ilustración 7 sin pérdida de información.

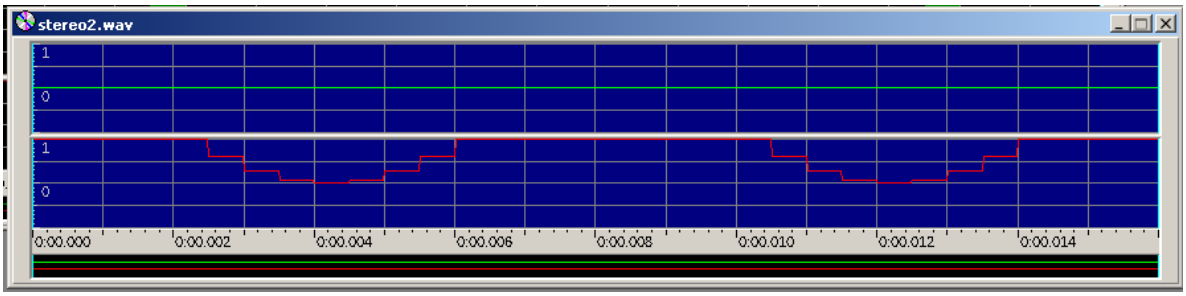

Ilustración 9 Resultado de elevar al cuadrado la señal de la ilustración 7 con pérdida de información

Conclusiones

Como podemos observar estamos limitados al tamaño máximo de un short y dependiendo que es lo que necesitemos es la implementación de escala que podemos ocupar, la multiplicación sirve para reconstruir una señal por lo cual al multiplicar en la frecuencia estaremos haciendo la convolución en el tiempo y seguramente necesitaremos la implementación con perdida para poder llegar a función original.

Código

Multiplicacion.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define PI 3.141592653589793
typedef struct CabeceraWAV{
	unsigned char ChunkID[4];
	unsigned int ChunkSize;
	unsigned char Format[4];
	unsigned char Subchunk1_ID[4];
	unsigned char Subchunk1_Size[4];
	unsigned char AudioFormat[2];
	unsigned char NumChannels[2];
	unsigned int SampleRate;
	unsigned int ByteRate;
	unsigned short BlockAlign;
	unsigned short BitsPerSample;
	unsigned char Subchunk2_ID[4];
	unsigned int Subchunk2_Size;
}Cabecera;

Cabecera getHeader(char *archivoWavEntrada){
	Cabecera CW;
	FILE *archivo=fopen(archivoWavEntrada,"r");
	fread(&CW,44,1,archivo);
	fclose(archivo);
	return CW;
}

//Resegra el tamanio completo de el archivo en bytes
unsigned int getFileSize(Cabecera CW){
	return CW.ChunkSize+8;
}

//Regresa el tamanio de las muestras en bytes
char getSampleSize(Cabecera CW){
	return (CW.BitsPerSample/8);
}

//Regresa el numero de canales;
char getCannalNumber(Cabecera CW){
	return CW.NumChannels[0];
}

//Regresa el numero de muestras de audio si es momo o estero regresa la
misma cantidad por lo cual en estereo se debe multiplicar por 2
unsigned int getNumberAudioSamples(Cabecera CW){
	return CW.Subchunk2_Size/(CW.NumChannels[0]*(CW.BitsPerSample/8));
}
//Regresa el numero de muestras de audio totales
unsigned int getTotalNumberSamples(Cabecera CW){
	return CW.Subchunk2_Size/(CW.BitsPerSample/8);
```

```
}

//Regresa el tamanio total del archivo wav-8
unsigned int getNumberBytesAudioInformation(Cabecera CW){
	return CW.Subchunk2_Size;
}

//Convierte una cabecera de tipo mono a una de tipo estereo
Cabecera setAudioStereo(Cabecera CW){
	CW.ChunkSize+=CW.Subchunk2_Size;
	CW.Subchunk2_Size*=2;
	CW.NumChannels[0]=2;
	CW.ByteRate*=2;
	CW.BlockAlign*=2;
	return CW;
}

//regresa el numero de bytes del pie del archivo wav
unsigned int getNumberBytesFoot(Cabecera CW){
	return CW.ChunkSize-36-CW.Subchunk2_Size;
}

//Regresa las muestras de un archivo con tamanio de 8 bits
char* getAudioSamples8bits(char *archivoWavEntrada){
	Cabecera CW;
	FILE *archivo=fopen(archivoWavEntrada,"r");
	fread(&CW,44,1,archivo);
	unsigned int
numeroMuestras=CW.Subchunk2_Size/(CW.NumChannels[0]*(CW.BitsPerSample/8))
;
	char *Samples=malloc(sizeof(char)*numeroMuestras);
	fread(Samples,numeroMuestras,1,archivo);
	fclose(archivo);
	return Samples;
}

//Regresa las muestras de audio de 16 bits mono/Stereo
short* getAudioSamples16bits(char *archivoWavEntrada){
	Cabecera CW;
	FILE *archivo=fopen(archivoWavEntrada,"r");
	fread(&CW,44,1,archivo);
	unsigned int
numeroMuestras=CW.Subchunk2_Size/(CW.NumChannels[0]*(CW.BitsPerSample/8))
;
	short
*Samples=malloc(sizeof(short)*numeroMuestras*CW.NumChannels[0]);
	fread(Samples,numeroMuestras*2*CW.NumChannels[0],1,archivo);
	fclose(archivo);
	return Samples;
}
short* getAudioSamples16bitsStereo(char *archivoWavEntrada){
	Cabecera CW;
	FILE *archivo=fopen(archivoWavEntrada,"r");
	fread(&CW,44,1,archivo);
	unsigned int
numeroMuestras=CW.Subchunk2_Size/(CW.NumChannels[0]*(CW.BitsPerSample/8))
;
```

```
	numeroMuestras*=2;
	short *Samples=malloc(sizeof(short)*numeroMuestras);
	fread(Samples,numeroMuestras*2,1,archivo);
	fclose(archivo);
	return Samples;
}

//Regresa las muestras de 16bits del canal1
short* getAudioSamples16bisCanal1(char *archivoWavEntrada){
	Cabecera CW=getHeader(archivoWavEntrada);
	if(getCannalNumber(CW)==2){
		short
*muestrasAudio=getAudioSamples16bitsStereo(archivoWavEntrada);
		unsigned int tam=getNumberAudioSamples(CW);
		short *canal1=malloc(tam*sizeof(short));
		for(unsigned int i=0;i<tam;i++){
			canal1[i]=muestrasAudio[2*i];
		}
		return canal1;
	}else{
		printf("El archivo: %s no es estereo\n",archivoWavEntrada);
		return NULL;
	}
}

//Regresa las muestras de 16bits del canal2
short* getAudioSamples16bisCanal2(char *archivoWavEntrada){
	Cabecera CW=getHeader(archivoWavEntrada);
	if(getCannalNumber(CW)==2){
		short
*muestrasAudio=getAudioSamples16bitsStereo(archivoWavEntrada);
		unsigned int tam=getNumberAudioSamples(CW);
		short *canal2=malloc(tam*sizeof(short));
		for(unsigned int i=0;i<tam;i++){
			canal2[i]=muestrasAudio[2*i+1];
		}
		return canal2;
	}else{
		printf("El archivo: %s no es estereo\n",archivoWavEntrada);
		return NULL;
	}
}

//Regresa las bytes del pie del archivo wav
char* getFileFoot(char *archivoWavEntrada){
	Cabecera CW;
	FILE *archivo=fopen(archivoWavEntrada,"r");
	fread(&CW,44,1,archivo);
	unsigned int bytesSobrantes=CW.ChunkSize-CW.Subchunk2_Size-36;
	char *bytes=malloc(sizeof(char)*CW.Subchunk2_Size);
	fread(bytes,CW.Subchunk2_Size,1,archivo);
	bytes=malloc(sizeof(char)*bytesSobrantes);
	fread(bytes,bytesSobrantes,1,archivo);
	return bytes;
}
short* getSalidaStereo(short *canal1,short *canal2,unsigned int
tamCanal){
```

```
	short *salida=malloc(sizeof(short)*tamCanal*2);
	for(unsigned int i=0;i<tamCanal;i++){
		salida[2*i]=canal1[i];
		salida[2*i+1]=canal2[i];
	}
	return salida;
}

char getTipoMultiplicacion(char wav1,char wav2){
	if(wav1==1){
		if(wav2==1)
			return 1;
		else
			return 2;
	}else{
		if(wav2==1)
			return 2;
		else
			return 3;
	}

}

double absDouble(double numero){
	if(numero<0)
		return numero*(-1);
	return numero;
}

int absNum(int numero){
	if(numero<0)
		return numero*(-1);
	return numero;
}

//muestras 1 es el arreglo mas grande
double* multiplicacionStereo(short *muestras1, short *muestras2, unsigned
int numMuestras1,unsigned int tope){
	double *mult=malloc(sizeof(double)*numMuestras1);
	for(unsigned int i=0; i<tope;i++){
		mult[i]=(double)(muestras1[i]*muestras2[i]);
		mult[i]=mult[i]/32767;
		if(mult[i]>32767)mult[i]=32767;
	}

	for(unsigned int i=tope;i<numMuestras1;i++)
		mult[i]=0;

	return mult;
}

short* multiplicacion(short *muestras1, short *muestras2, unsigned int
numMuestras1,unsigned int tope){
	double *mult=malloc(sizeof(double)*numMuestras1);

	for(unsigned int i=0; i<tope;i++){
		mult[i]=(double)(muestras1[i]*muestras2[i]);
```

```
			mult[i]=mult[i]/32767;
			if(mult[i]>32767)mult[i]=32767;
			muestras1[i]=mult[i];

		}
		for(unsigned int i=tope;i<numMuestras1;i++){
			mult[i]=0;
		}
		return muestras1;
}

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

		FILE *archivoWavEntrada1,*archivoWavEntrada2,*archivoWavSalida;
		archivoWavEntrada1=fopen(argv[1],"r");
		archivoWavEntrada2=fopen(argv[2],"r");

		if(archivoWavEntrada1==NULL || archivoWavEntrada2==NULL){
			printf("Error en la lectura de los archivos de audio\n");
		}else{
			char *archivo1=argv[1];
			char *archivo2=argv[2];
			Cabecera CW1=getHeader(archivo1);
			Cabecera CW2=getHeader(archivo2);
			char tipoWav1=getCannalNumber(CW1);
			char tipoWav2=getCannalNumber(CW2);
			char tipoMult=getTipoMultiplicacion(tipoWav1,tipoWav2);

			if(tipoMult==2){//archivo 1 siempre debe ser el monoaural
				if(getCannalNumber(CW2)==1){
					char *aux=archivo1;
					archivo1=archivo2;
					archivo2=aux;
				}
			}
			CW1=getHeader(archivo1);
			CW2=getHeader(archivo2);
			archivoWavSalida=fopen(argv[3],"w");

			if(archivoWavSalida!=NULL){
					int numMuestrasAudio1=getTotalNumberSamples(CW1);
					int numMuestrasAudio2=getTotalNumberSamples(CW2);
					short *muestras1=getAudioSamples16bits(archivo1);
					short *muestras2=getAudioSamples16bits(archivo2);


					switch(tipoMult){
						case 1:// Mono * Mono

		if(numMuestrasAudio1<=numMuestrasAudio2){

		fwrite(&CW2,44,1,archivoWavSalida);

		fwrite(multiplicacion(muestras2,muestras1,numMuestrasAudio2,numMues
trasAudio1),getNumberBytesAudioInformation(CW2),1,archivoWavSalida);
					char *pie=getFileFoot(archivo2);
```

```
      fwrite(pie,getNumberBytesFoot(CW2),1,archivoWavSalida);
                                    }
                                    else{

      fwrite(&CW1,44,1,archivoWavSalida);

      fwrite(multiplicacion(muestras1,muestras2,numMuestrasAudio1,numMues
trasAudio2),getNumberBytesAudioInformation(CW1),1,archivoWavSalida);
                                          char *pie=getFileFoot(archivo1);

      fwrite(pie,getNumberBytesFoot(CW1),1,archivoWavSalida);
                                    }

                                    break;
                              case 2://Mono * Stereo

      if(numMuestrasAudio1<(numMuestrasAudio2/2)){

      fwrite(&CW2,44,1,archivoWavSalida);
                                                numMuestrasAudio2/=2;
                                                short
*canal1=multiplicacion(getAudioSamples16bisCanal1(archivo2),muestras1,num
MuestrasAudio2,numMuestrasAudio1);
                                                short
*canal2=multiplicacion(getAudioSamples16bisCanal2(archivo2),muestras1,num
MuestrasAudio2,numMuestrasAudio1);

      fwrite(getSalidaStereo(canal1,canal2,numMuestrasAudio2),getNumberBy
tesAudioInformation(CW2),1,archivoWavSalida);
                                                char
*pie=getFileFoot(archivo1);

      fwrite(pie,getNumberBytesFoot(CW2),1,archivoWavSalida);
                                          }else{
                                                CW1=setAudioStereo(CW1);

      fwrite(&CW1,44,1,archivoWavSalida);
                                                numMuestrasAudio2/=2;
                                                short
*canal1=multiplicacion(muestras1,getAudioSamples16bisCanal1(archivo2),num
MuestrasAudio1,numMuestrasAudio2);
                                                short
*canal2=multiplicacion(getAudioSamples16bits(archivo1),getAudioSamples16b
isCanal2(archivo2),numMuestrasAudio1,numMuestrasAudio2);

      fwrite(getSalidaStereo(canal1,canal2,numMuestrasAudio1),getNumberBy
tesAudioInformation(CW1),1,archivoWavSalida);
                                                char
*pie=getFileFoot(archivo2);

      fwrite(pie,getNumberBytesFoot(CW2),1,archivoWavSalida);
                                          }

                                    break;
                              case 3:// Stereo * Stereo
```

```
      if(numMuestrasAudio1<=numMuestrasAudio2){

      fwrite(&CW2,44,1,archivoWavSalida);
                                          numMuestrasAudio2/=2;
                                          numMuestrasAudio1/=2;
                                          double
*op1=multiplicacionStereo(getAudioSamples16bisCanal1(archivo2),getAudioSa
mples16bisCanal1(archivo1),numMuestrasAudio2,numMuestrasAudio1);
                                          double
*op2=multiplicacionStereo(getAudioSamples16bisCanal2(archivo2),getAudioSa
mples16bisCanal2(archivo1),numMuestrasAudio2,numMuestrasAudio1);
                                          double
*op3=multiplicacionStereo(getAudioSamples16bisCanal2(archivo2),getAudioSa
mples16bisCanal1(archivo1),numMuestrasAudio2,numMuestrasAudio1);
                                          double
*op4=multiplicacionStereo(getAudioSamples16bisCanal1(archivo2),getAudioSa
mples16bisCanal2(archivo1),numMuestrasAudio2,numMuestrasAudio1);

                                          double
*c1=malloc(sizeof(double)*numMuestrasAudio2);
                                          double
*c2=malloc(sizeof(double)*numMuestrasAudio2);
                                          short
*canal1=malloc(sizeof(short)*numMuestrasAudio2);
                                          short
*canal2=malloc(sizeof(short)*numMuestrasAudio2);

                                          for(unsigned int
i=0;i<numMuestrasAudio2;i++){
                                                c1[i]=(op1[i]-
op2[i])/2;//Multplicacion sin perdida
                                                c2[i]=(op3[i]+op4[i])/2;//
                                          /*    c1[i]=(op1[i]-
op2[i]);//////////////
                                                c2[i]=(op3[i]+op4[i]);

      if(c1[i]>32767)c1[i]=32767;

      if(c2[i]>32767)c2[i]=32767;
                                                if(c1[i]<-32768)c1[i]=-
32768;
                                                if(c2[i]<-32768)c2[i]=-
32768;///////////////Multiplicacion con perdida*/
                                                canal1[i]=c1[i];
                                                canal2[i]=c2[i];
                                          }

      fwrite(getSalidaStereo(canal1,canal2,numMuestrasAudio2),getNumberBy
tesAudioInformation(CW2),1,archivoWavSalida);
                                          char *pie=getFileFoot(archivo2);

      fwrite(pie,getNumberBytesFoot(CW2),1,archivoWavSalida);
                                    }
                                    else{

      fwrite(&CW1,44,1,archivoWavSalida);
```

```
                                        numMuestrasAudio2/=2;
                                        numMuestrasAudio1/=2;
                                        double
*op1=multiplicacionStereo(getAudioSamples16bisCanal1(archivo1),getAudioSa
mples16bisCanal1(archivo2),numMuestrasAudio1,numMuestrasAudio2);
                                        double
*op2=multiplicacionStereo(getAudioSamples16bisCanal2(archivo1),getAudioSa
mples16bisCanal2(archivo2),numMuestrasAudio1,numMuestrasAudio2);
                                        double
*op3=multiplicacionStereo(getAudioSamples16bisCanal1(archivo1),getAudioSa
mples16bisCanal2(archivo2),numMuestrasAudio1,numMuestrasAudio2);
                                        double
*op4=multiplicacionStereo(getAudioSamples16bisCanal2(archivo1),getAudioSa
mples16bisCanal1(archivo2),numMuestrasAudio1,numMuestrasAudio2);


                                        double
*c1=malloc(sizeof(double)*numMuestrasAudio1);
                                        double
*c2=malloc(sizeof(double)*numMuestrasAudio1);
                                        short
*canal1=malloc(sizeof(short)*numMuestrasAudio1);
                                        short
*canal2=malloc(sizeof(short)*numMuestrasAudio1);

                                        for(unsigned int
i=0;i<numMuestrasAudio2;i++){
                                              c1[i]=(op1[i]-op2[i])/2;
                                              c2[i]=(op3[i]+op4[i])/2;
                                        /*    c1[i]=(op1[i]-
op2[i]);//////////////
                                              c2[i]=(op3[i]+op4[i]);

      if(c1[i]>32767)c1[i]=32767;

      if(c2[i]>32767)c2[i]=32767;
                                              if(c1[i]<-32768)c1[i]=-
32768;
                                              if(c2[i]<-32768)c2[i]=-
32768;////////////////*/
                                              canal1[i]=c1[i];
                                              canal2[i]=c2[i];
                                        }


      fwrite(getSalidaStereo(canal1,canal2,numMuestrasAudio1),getNumberBy
tesAudioInformation(CW1),1,archivoWavSalida);
                                        char *pie=getFileFoot(archivo1);

      fwrite(pie,getNumberBytesFoot(CW1),1,archivoWavSalida);
                                  }
                                  break;
                            default:
                                  printf("Error\n");
                      }
                      fclose(archivoWavSalida);
                      fclose(archivoWavEntrada1);
```

```
                        fclose(archivoWavEntrada2);
                              return 0;
                        }else{
                              printf("No se pudo crear el archivo de
salida\n");
                              return 0;

                  }
            }
            return 0;

}
```