ECE 3331, Dr. Hebert, Fall 2023, Programming Assignment 13 due Saturday 11/25 at 11:59 pm

File ts9_mono is a short wav file (mono, 16 bits per sample, samplerate = 22050/sec) containing a simple recorded guitar track. You will use the audio data in this mono WAV file to make a stereo WAV file that has an flange-effect applied to the guitar sound. Flange is a digital audio effect. The file ts9_stereo_flange is an example of what your output file should sound like when it is correct.
In this project you will use malloc( ) and a 1-D array of structures.

To start your project,
(1) Above your main ( ) function in your program file after the preprocessing commands, declare a global WAV header structure containing all the required fields for a WAV file header. This new data type will then be used in a subroutine in bot hits function declaration and function decription. As in lecture, you can use something like

```
struct wavheader{
  char chunkid[4];
  unsigned int chunksize;
  char format[4];
  char subchunk1id[4];
  unsigned int subchunk1size;
  unsigned short int audioformat;
  unsigned short int numchannels;
  unsigned int samplerate;
  unsigned int byterate;
  unsigned short int blockalign;
  unsigned short int bitspersample;
  char subchunk2id[4];
  unsigned int subchunk2size;
};
```

(2) Write a **function declaration (**you choose the function name, no return value**)** that takes as its arguments a pntr-to-pntr-to-FILE (FILE **) and a pointer-to a WAV header structure (struct wavheader *). This function will be used to open an **input** WAV file and read-in a WAV header structure from the opened file, storing the pointer-to-FILE and the WAV header structure in the main( ) function variables. The subroutine need to receive a pntr-to-pntr-to-FILE because the subroutine will open the file, thus changing the FILE pointer in the main( ) function so that it points to a particular file. So the subroutine needs to change the FILE pointer that resides in the main( ) function. You will write the function definition in a later step. Here in step (2), you only need to write the function declaration.
(3) Write a 2nd **function declaration (**you choose the name**)** take a pointer-to-pointer-to-FILE as its argument and returns no value (void). This function will be used to open an **output** WAV file. Again, here in step (3), you only need to write the function declaration.
(4) Begin your main( ) function. Declare two FILE pointers, one for an input WAV file and one for an output WAV file. Declare two struct headers using a two-element 1-D array of the new struct data type defined as the WAV header structure. The 0th element of the 1-D array will be the header for an input WAV file and the 1th element will be the header for the output WAV file.
Next, your main( ) function should call the subroutine declared in step (2) above to open an input WAV file and read its header. The function will prompt the user for the filename. In your main( ) function, write a statement to call the subroutine declared in step (2), using as the subroutine argument the pointer to the 0th element of the WAV header structure vector. This subroutine will also initalize the FILE pointer from your main function to point the input file, so that your program can access the audio data in the input file.
(5) Now it is time to write the function definition for the subroutine in step (4) above. This function will prompt the user for the name of an input wave file, open the input WAV file, read the header, and initialize the FILE pointer for the opened file (without closing the file). If you have any questions here, this subroutine is similar to that from a previous programming assignment.
Again, this subroutine takes as its arguments a pointer-to-pointer-to-FILE and a **pointer to a WAV header structure.** Open the user-provided filename in mode "rb" using the FILE pointer. Read the header into the memory location pointed to by the pointer to the structure header received as an argument in the function. Use a single fread( ) to read-in the header. Check that the input WAV file is both 16 bitspersample and a mono WAV file. If the file doesn't exist, or if it has

the wrong WAV characteristics, close the file and prompt the user to enter a correct filename. Repeat this until a valid filename is provided and the file is successfully opened.

Before moving to the next step, now is a good time to check how your program is working so far.

(6) Turn your attention back to your main( ) function. In your main( ) function, after the subroutine call in step (4), your main( ) function should call the subroutine declared in step (3) to get an **output** WAV file pointer. Write a single statement in your main( ) function to call this subroutine.

(7) Now go below your main( ) function, and write the function definition for the subroutine declared in step (3). This subroutine will prompt the user for a path and filename for an **output** WAV file. Your subroutine will open this WAV file in mode "wb".

Now is a good time to again check how your program is working so far.

(8) Turn your attention back to your main( ) function. In the main( ) function, generate the output **stereo WAV header** in the 1th element of the WAV header structure vector. It should have the same characteristics as the input mono wav file header, except that i twill be stereo - not mono, nd the size of the audio data willl be twice that of the input WAV mono file. You could initialize the header for the output WAV file by 1 setting the 1th element of your WAV header structure vector (output WAV header) to the 0th element value (input WAV header) ; then change only he fileds that need changing. For example if your 2-element WAV header vector is wh[.]; simply use wh[1]=wh[0], then you only need to change the appropriate fields of wh[1].
Change the values for filesize (chunksize),
change the number of channels to 2,
change the data subchunksize to twice its size,
change the byterate to twice the byterate of the mono file.,
change the blockalign value.
Did we leave anything out? Think through the complete process again.

(9) In your main( ) function, malloc a short int (16-bit) vector to hold the audio data from the input mono wav file, and read the audio data from the input file into the malloc'd vector using a single fread( ) statement. Now malloc two vectors of the same size to hold the output data for the stereo file, one for the left channel and one for the right channel.

(10) Let's again check if your code so far is working. In your main( ) function, write statements to copy the input **mono data** directly into the malloc'd vectors for the left channel and the right channel, then write out a stereo WAV file using the output WAV header you created in step (8). That output WAV file should play. To do this, first, fwrite( ) the output WAV header in the 1th element of the WAV header structure vector to the output file, then use the left and right channel vectors to write the audio data. Do this with a for( ) loop; fwrite( ) a left channel sample, then a right channel sample; etc. Close the output file.
Now compile and run your program using **ts9_mono.wav** as the input file. It contains a simple recorded guitar.

Check to be sure the output stereo WAV file plays.

Play the file that your program created and also verify its size. The output stereo WAV file should play, and it should be almost twice the size of the input wav file. If the output WAV file does not play, debug your program. Once your stereo file plays correctly, go to the next step.

(11) Now you will change your program so that it adds a flange effect to the input audio data to create the output stereo data.
Your program will process the input audio data in a loop to create the left channel data and the right channel data for the output file.

A flange effect is created by adding a delayed version of the signal to itself, and varying the amount of delay.
If the nth input audio sample and the nth output audio sample are denoted $ai[n]$ and $ao[n]$ a fixed delay version of an audio signal is added to itself by
$ao[n] = ai[n] + ai[n-d]$.
Since the sum $ao[n]$ might exceed the range of a short int (for example) , we need to divide by 2.0

$$ao[n] = \frac{ai[n] + ai[n-d]}{2}.$$

If, for example, the sample rate is 22050 samples per second and we want an echo with a 0.002 [secs] delay, then d= 0.002*22050 = 44.

$$ao[n] = \frac{ai[n] + ai[n-44]}{2}.$$

Here, the echo is the same volume as the direct sound.
We can decrease the volume of the "echo" $ai[n-d]$ for example via

$$ao[n]=\frac{ai[n]+0.7*ai[n-44]}{1.0+0.7}.$$

The flange effect is formed by a time-varying delay *d[n]*

$$ao[n]=\frac{ai[n]+0.7*ai[\,n-d[n]]}{1.0+0.7}$$ where, (continuing with the example with sample rate = 22050 , delay = 0.002 secs, and delay in samples d= 0.002*22050 = 44).

$d[n]=\text{int}\left(44\left[1+A\cos\left(\frac{2\pi f_0 n}{f_s}\right)\right]\right)$ and f_s is the samples per second of the audio file, f_0 is a small frequency such as 0.8 Hz, and amplitude A equals a value less than or equal to 1.0; for example A=0.9.
In this example, the output equation would be

$$ao[n]=\frac{ai[n]+0.7*ai\left[n-\text{int}\left(44\left[1+0.9\cos\left(\frac{2\pi f_0 n}{f_s}\right)\right]\right)\right]}{1.0+0.7}.$$

(You will have to adjust the value 44 for the actual samplerate.)

To create a stereo flange effect, try using a cosine function to create the left channel data from the input mono data, and a sine wav for to create the right channel data

$$ao_L[n]=\frac{ai[n]+0.7*ai\left[n-\text{int}\left(44\left[1+0.9\cos\left(\frac{2\pi 0.8n}{f_s}\right)\right]\right)\right]}{1.0+0.7}$$

$$ao_R[n]=\frac{ai[n]+0.7*ai\left[n-\text{int}\left(44\left[1+0.9\sin\left(\frac{2\pi 0.8n}{f_s}\right)\right]\right)\right]}{1.0+0.7}$$

(You will have to adjust the value 44 for the actual samplerate.)
delay = 0.003 secs samplerate=44100 yields delay in samples d= (int)(0.003*44100) = 132).

At the start of this algorithm (n=0, 1, 2, etc) the index into the 1-D array $n-\text{int}\left(44\left[1+0.9\sin\left(\frac{2\pi 0.8n}{f_s}\right)\right]\right)$ may be less than 0,

so that $ai\left[n-\text{int}\left(44\left[1+0.9\sin\left(\frac{2\pi 0.8n}{f_s}\right)\right]\right)\right]$ may not exist (i.e. ai[-4] doesn't exist). If

$n-\text{int}\left(44\left[1+0.9\sin\left(\frac{2\pi 0.8n}{f_s}\right)\right]\right)<0$,

use ai[n] in place of $ai\left[n-\text{int}\left(44\left[1+0.9\sin\left(\frac{2\pi 0.8n}{f_s}\right)\right]\right)\right]$.

(12) Once your program is working and creating a playable wav file, you can experiment with different values of amplitude A between 0 and 1.0 and different values of f_0 between 0.2 and 1.2. However, the provided values are close to the best sounding values.