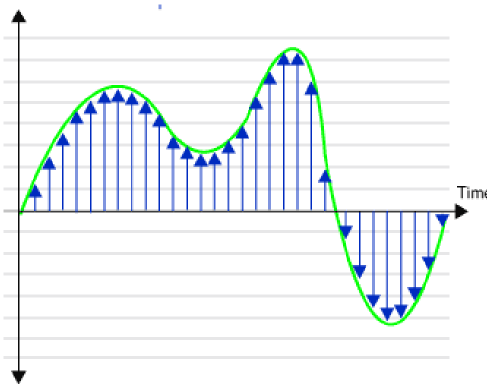
Goal:

Manipulation of WAV audio files:
(1) Implement a function to adjust the tone (bass vs. treble) of an audio recording.

Objective: Familiarity with the WAV audio library (libwav.a).

Background: Waveform Audio File Format (more commonly known as WAV due to its filename extension) is a Microsoft file format standard for storing an audio recording. WAV files store a digital representation of sound by measuring the signal amplitude at regular intervals and recording the sampled value as an integer. Two basic properties determine quality: the sampling rate, which is the number of times per second that samples are taken; and the bit depth, which determines the number of possible digital values that can be used to represent each sample. This assignment uses single-channel (monophonic) WAV files that store sampled audio as a one-dimensional array of 16-bit signed integers.



Download: Download and unpack file lab8.zip from Camino.

WAV Library: Audio recordings are stored in memory using the following data structure:

```
typedef int16_t SAMPLE ;

typedef struct
        {
        unsigned      sample_rate ;
        unsigned      num_samples ;
        SAMPLE        samples[] ;
        } AUDIO ;
```

The library file (libwav.a) provides several functions needed for the manipulation of WAV files:

`AUDIO *`**`NewAudio`**`(unsigned` *samples*`, unsigned` *rate*`) ;`
Returns a pointer to memory allocated to hold an audio recording in which the number of samples is specified by parameter *samples* and sampled a number of times per second specified by parameter *rate*. The actual sample values, however, are not initialized and must be replaced by the user.

`AUDIO *`**`ReadWAV16`**`(char *`*filespec*`) ;`
Allocates memory to hold an image and fills it from a 16-bit monophonic WAV file. Parameter *filespec* is the name of the file given as a character string. The return value is a pointer to the audio in memory and must be used as an argument to all audio manipulation functions.
`void `**`WriteWAV16`**`(char *`*filespec*`, AUDIO *`*audio*`) ;`

Stores an audio recording from memory into a 16-bit monophonic WAV file. Parameter *filespec* is the filename specified as a character string.

```
void FreeAudio(AUDIO *audio) ;
```

Releases the memory used by an audio recording.

```
AUDIO *CopySegment(AUDIO *source, unsigned frstndex, unsigned lastndex) ;
```

Copies an audio segment from *source* starting at index position *frstndex*, through and including index position *lastndex*. Returns a pointer to a new memory representation of the copy.

```
AUDIO *InsertSegment(AUDIO *target, AUDIO *segment, unsigned at) ;
```

Inserts an audio segment specified by *segment* into the memory representation of *target* beginning at the index position specified by parameter *at*. Returns a pointer to the modified (lengthened) memory representation of *target*.

```
AUDIO  *DeleteSegment(AUDIO * source, unsigned frstndex, unsigned lastndex) ;
```

Deletes an audio segment from *source* between index positions *frstndex* through *lastndex*. Returns a pointer to the modified (shortened) memory representation of *source*.

Assignment: You are to complete the source code for the following function that is located within the provided main program (lab8.c):

```
AUDIO *AdjustTone(AUDIO *audio, unsigned percent_bass, unsigned percent_treble);
```

Adjusts the tone of an audio recording by adjusting the relative amount of bass and treble.

Parameters *percent_bass* and *percent_treble* are unsigned integers in the range 0 to 100.

The apparent amount of ***bass*** (low frequencies) in a recording can be increased by using a running weighted average to smooth out the audio waveform and the apparent amount of ***treble*** (high frequencies) can be increased using the difference between successive samples. The pseudo-code shown below computes values for the bass and treble components and then mixes them with the original sample to produce the new sample value.

```
sample_avg   ← 0.9 × sample_avg + 0.1 × orig_sample
sample_diff  ← (orig_sample – prev_sample)

bass_part    ← sample_avg  × (percent_bass / 100)
orig_ part   ← orig_sample × (100 – percent_bass – percent_treb) / 100)
treb_ part   ← sample_diff × (percent_treb / 100)

prev_sample  ← orig_sample

curr_sample  ← 2 × bass_ part + orig_ part + 2 × treb_ part
```

Note: The values of *sample_avg* and *prev_sample* should be initialized to 0.

Compilation: Compile and link your program using the following command line:

gcc –o lab8 lab8.c -L. -lwav

Execution: Execute your program using the following command syntax:

./lab8 *src-file dst-file tone-knob-degrees*

where *tone-knob-degrees* is an integer between -90 (full bass) and +90 (full treble); the program uses this value to compute values for *percent_bass* and *percent_treb*.

When Done: Demonstrate proper operation of your program to the teaching assistant and upload the completed source code for file lab8.c to the lab drop box on Camino. Do not upload any other files.