Problem Set 5: Reading and Writing WAV Files

Please send back via NYU Classes

- A zip archive named as
  PS05_<your name as Last_First>.zip
  containing the C code files that implement all aspects of all problems.

Total points: 100

Points are awarded as follows:

- **20 Points** - parse comment line and print usage
- **20 Points** - open WAV, read and parse header and print indicated information
- **20 points** - read binary WAV information and calculate max abs value
- **10 points** - adjust max abs value of WAV signal per command line amplitude
- **10 points** - write WAV header and WAV signal to file and close file
- **20 Points** - clear code, sensible formatting, good comments

Read and write WAV file

wav_io.c

Create a program that reads a WAV file, adjusts the maximum absolute value of the amplitude of the signal, and saves the adjusted data in a new WAV file. The bullet items below highlight the components of the program that you must include.

- The program should use the following command line:

```
./wav_io max_amplitude ifile ofile
```

where

`max_amplitude` - the maximum absolute level in ofile on the scale of 0 to 32767
`ifile` - the input audio file in *.wav format
`ofile` - the output audio file in *.wav format

Consider using `atoi()` to convert the arg string `max_amplitude` to a integer value. This requires `#include <stdlib.h>`.

- If the correct arguments are not present, the program should print a usage statement.

Binary file I/O

Assume that `ifile` and `ofile` are single channel, 16-bit PCM WAV format. Use the supplied file **tone1k.wav** as ifile. Assume the simplest structure of a WAV file: that it has

a 44-byte header followed by a single chunk of audio signal data. More information about the WAV header is at: http://soundfile.sapp.org/doc/WaveFormat/
You could review **Big-Endian** and **Little-Endian** data storage formats, but you will write the program in a way that Endian-ness is of (almost) no concern.

- Open `ifile` for binary reading, using error checking and reporting.

- Read 44 bytes (the WAV header) into an unsigned char array, reporting errors:
```
unsigned char header[44];
if ( fread(header, sizeof(header), 1, ifp) == 0 ) {
     // error reporting
}
```

- Print each byte of the wav header as both character and hexadecimal values with 4 lines of 11 characters per line. If `header[i]` is a printable character, then print the character, otherwise print a SPACE character. You might use this code to detect printable characters and then to print the character:
```
c = isprint(header[i]) ? header[i] : ' ';
printf("%c %02x ", c, header[i]);
```

You will need to use `#include <ctype.h>`. Confirm that this is a valid WAV header by visual inspection with respect to the information provided at http://soundfile.sapp.org/doc/WaveFormat/

- Print the following information about the WAV file:
  - Number of channels
  - Sampling Rate
  - Bits per audio sample
  - The data size, in bytes, of the sound data portion of the WAV file (this is "Subchunk2Size" in writeup at the above URL)

Note that header[] is a unsigned char array and the values to be printed (listed above) are 16 or 32 bit integer values. Also, note that they are Little-Endian in their byte layout. You can construct the 16-bit value NumChannels from the header bytes as:
```
NumChannels = header[23]<<8 | header[22];
```

You can construct a 32-bit value SampleRate from the header bytes as:
```
SampleRate = header[27]<<24 | header[26]<<16 | header[25]<<8 | header[24];
```

Note that reading the header as unsigned char (i.e. bytes) circumvents the entire Endian-ness issue. Calculating 16 or 32 bit values from the header bytes is where Endian-ness comes into play. The WAV file has Little-Endian data organization.

Confirm that the data size is consistent with the file size (i.e. file size = 44 + data size) by running the following command in the terminal:

```
ls –l tone1k.wav
```

Binary calculations

Your program must calculate the max absolute value of the audio data signal. Determine the number of samples in `ifile` by using the data size and bits per sample from the WAV header.

- Declare a pointer to short (i.e. 16-bit PCM) to hold all of the ifile values and allocate storage to the pointer using `malloc()` with error checking and reporting.

```
short *x;
x = (short *)malloc(num_samp * sizeof(*x));
if ( x == NULL) {
     // error reporting
}
```

- Read all of the values into `x` using `fread()` with error checking and reporting.

```
count = fread(x, sizeof(*x), num_samp, ifp)
if ( count != num_samp ) {
     // error reporting
}
```

Note that when you open a file with fopen(), the first byte read with fread() is the first byte in the file. However, the FILE pointer remembers "where you are" in the file in terms of reading. Hence, after you read the 44-byte WAV header, the next byte read by fread() will be the audio data.

Further note that the WAV file is Little-Endian data, and the Intel platform is Little-Endian, so reading a short from the file is compatible with a short variable on the Intel platform.

The maximum absolute value can be calculated as follows:

```
int max_x = 0;
for (i=0; i<num_samp; i++) {
     if (abs(x[i]) > max_x) {
          max_x = abs(x[i]);
     }
}
```

- Print the max abs value.

Adjust the array x and write to output WAV file

- Modify the values of x so that the max absolute value is equal to what is requested in the command line. Do this by multiplying every value of x as

```
x[i] = (short)(x[i] * ((float)max_amplitude/(float)max_x) + 0.5);
```

where `max_amplitude` is the value given on the command line and `max_x` is the calculated max abs value as shown above. Since these are integers, be sure to cast them to float when doing the division. After division, you want to round to the nearest short value. Since assigning a float to a short results in dropping the fractional part, adding the 0.5 value before assignment effectively results in rounding to the nearest integer value.

As shown above, you can overwrite the x array with the scaled `x` values).

- Write the adjusted `x` to `ofile` with a WAV header. Since the length and format of `ofile` is identical to that of `ifile`, you can write out the exact same header that you read in. Since the header is at the front (head) of the output file, followed by the PCM data, first write the header[] array and then the x[] array. Both writes are binary data, so use `fwrite()`.

Verify that the output file is still a WAV file (by playing it) and that it seems to have the correct amplitude.