Problem Set 5: Play WAV Files

Please send back to me via NYU Brightspace

- A zip archive named as
  `PS05_First_Last.zip`
  Where First and Last are your first and last names.
  containing the C code files that implements all aspects of all problems.

Total points: 100

Problem 1
Points are awarded as follows:

- **20 Points** - parse command line, print usage or open file, print audio signal information, close file.
- **20 Points** – Port Audio callback code
- **10 Points** - clear code, sensible formatting, good comments

Problem 2
Points are awarded as follows

- **20 Points** - parse command line, print usage or open file, print audio signal information, malloc buffer for file, read file, close file.
- **20 Points** – Port Audio callback code
- **10 Points** - clear code, sensible formatting, good comments

References
Wav file reference
http://soundfile.sapp.org/doc/WaveFormat/

libportaudio reference
http://www.portaudio.com/

libsndfile reference
http://www.mega-nerd.com/libsndfile/

The following URLs provides a good reference for C language library function usage:
https://en.cppreference.com/w/
http://www.cplusplus.com/reference/cstdlib/

Overview
Add to the code in the instructor-supplied file **play_wavfiles_sndfile.c** and **play_wavfiles_ptr.c** to fill in code under the comment `//Your Code Here`.

You are given Bash script **build.sh** that compiles and links this program on all platforms.

Your program plays WAV file to the laptop speaker. Program uses:

sndfile library to read WAV files

PortAudio library to play audio data to speaker using a callback function

libsndfile, libportaudio

You should have these installed these libraries at the first lab session.

Problem 1

This version of the program uses calls to `sf_readf_float()` in the Port Audio callback to read audio samples directly into the callback output buffer.

This is NOT the "correct" way to handle audio data in a callback. Calls to `sf_readf_float()` in the Port Audio callback will in turn call `fread()` "system" code, which may block or wait for the operating system to permit it to read from disk. This could result in a real-time fault in the callback (an audio "dropout"). However, in my experience this always works (but you still should not do it this way!)

Your program should have the following command line usage:

- `./a.out ifile.wav`

Where

- `ifile.wav` is the WAV audio files to be played

Your program should

- Parse the command line. If parsing fails, print an error diagnostic and exit.
- Open the WAV file using `sf_open()`, which reads the WAV header into the `sndfile` structure.
- Print information about the WAV file.
- If the number of channels > MAX_CHN then print an error message and exit.
- Start PortAudio

In the Port Audio callback

- If p->play is 0 then copy zeros to the callback output buffer
- Otherwise, call `sf_readf_float()` to read `framesPerBuffer` audio frames into the callback `output` buffer. If the end of the audio file is reached (if the value `count` returned from `sf_readf_float()` is less than `framesPerBuffer`, then call `sf_seek()` to rewind the WAV file to the beginning and read the remaining `(framesPerBuffer-count)` frames into the output buffer so that playout of the file loops.

When user enters 'CR' the playout should toggle between play and pause

When user enters 'Q' the program should

- Stop PortAudio
- Close the WAV file

Problem 2

In this version of the program the main() routine opens the WAV file, allocates sufficient memory to read the file into a buffer, reads the audio data into the buffer and initializes pointers in the Port Audio callback data structure that point into the buffer. The pointers point to the next audio sample to read, the bottom of the buffer and top of the buffer. The Port Audio callback uses the pointers to copy audio samples the buffer to the callback output buffer.

This is the "correct" way to handle audio data in a callback. There is never a call to "system" code in the callback, but rather only to user code which you (the programmer) wrote and know will never block operation.

Your program should have the following command line usage:

- `./a.out ifile.wav`

Where

- `ifile.wav` is the WAV audio files to be played

Your program should

- Parse the command line. If parsing fails, print an error diagnostic and exit.
- Open the WAV file using `sf_open()`, which reads the WAV header into the `sndfile` structure.
- Print information about the WAV file.
- If the number of channels > MAX_CHN then print an error message and exit.
- Allocate storage sufficient to read the entire WAV file audio signal.
- Read the WAV file audio signal into the buffer.
- Close the WAV file.
- Initialize the pointers in the WAV data structure
- Start PortAudio

In the Port Audio callback

- If p->play is 0 then copy zeros to the callback output buffer
- Otherwise, copy, using the `p->next` pointer, `framesPerBuffer` audio frames from the buffer allocated in main() to the callback `output` buffer. For every sample copied, check if `p->next` is incremented past the end of the buffer, i.e., if (`p->next > p->top`). If so, then set `p->next = p->bottom` so that the audio playout loops.

When user enters 'CR' the playout should toggle between play and pause
When user enters 'Q' the program should

- Stop PortAudio