Problem Set 7: Play WAV Files

Please send back to me via NYU Brightspace

- A zip archive named as
  `PS07_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

Points are awarded as follows:

- **50 Points** - parse command line, print usage or open files, save audio information, allocate buffer for file, read file into buffer, close file.
- **30 Points** - Play file to audio output
- **20 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.c** and fill in code under the comment `//Your Code Here`. You are also given `ifile_list.txt` and all of the WAV files listed in that file (in the `signals` folder).

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

Your program plays a selected one of several WAV files to the laptop speaker.
Program uses:

sndfile library to read WAV files
ncurses library to read key presses without waiting for CR
PortAudio library to play audio data to speaker using a callback function

libsndfile, libportaudio, Ncurses
You should have these installed these libraries at the first lab session.

Program Overview

Your main program will

- Open a set of WAV files
- Check that they are consistent (same number of channels and same sampling rate)
- Read the audio data from each file into its own buffer
- Start PortAudio
- Create a simple Ncurses user interface to select which file to play

Your PortAudio callback will

- Copy `FRAMES_PER_BUFFER` audio frames from the audio file buffer associated with the selected filename to the callback `output` buffer. If the end of the audio file buffer is reached, reset the `next_frame` variable to zero so that play loops back to the start of audio.

(50 points) Parse command line, open list files, process each WAV file

Your program should hav the following command line usage:

```
./a.out ifile_list.txt
```

Where

`ifile_list.txt` is a plain text file that contains a list of WAV audio files that can be played

Parse command line and open input text file

Parse the command line. If parsing fails, print an error diagnostic and exit. If successful, open `ifile_list.txt` that is the list of WAV files.

Loop over each input file in the list

Use a loop to read each line of the file `ifile_list.txt`:

```
for (i=0; i<MAX_FILES; i++) {
…
}
```

Inside this loop you program will:

- Read the WAV filename from the input file.
- Open the WAV file using sf_open(), which reads the WAV header into the `sndfile` structure.
- Print information about the WAV file.
- Write selected information to the PortAudio callback `buf` structure.
- Check that
    - The number of channels in the WAV file is not more than MAX_CHAN
    - All files have the same number of channels and the same sampling rate.
- Allocate storage sufficient to read the entire WAV file audio signal.

- Read the WAV file audio signal into the buffer.
- Close the WAV file.
- Write the necessary information for this file into the WAV data structure
- Count how many WAV files were processed (were in the list).

Details on each step are given below.

Use `scanf()` to read each line:

```
if (fscanf(fp, "%s", ifilename[i]) == EOF)
    break;
```

where `fp` is the FILE pointer from opening `ifile_list.txt`. This works since there is only one string in each line of the file. Save the filename in an array of strings, ifilename[i], so that they can be displayed in the Ncurses loop.

Use the libsndfile library to open WAV audio files and read the WAV header. Use error checking and error reporting in all operations. The libsndfile data structures are declared at the top of main():

```
SNDFILE *sndfile;
SF_INFO sfinfo;
```

Your program will re-use the structures for each input WAV file.

Open each WAV file open using `sf_open()`

```
sndfile = sf_open (ifilename[i], SFM_READ, &sfinfo))
```

where

`sndfile` is the `SNDFILE` structure,
`ifilename[i]` is array of WAV filenames and
`sfinfo` is the `SF_INFO` structure..

if `sf_open()` returns `NULL`, then this is a file open error so print an error message and return with `-1`.

Members of the SF_INFO structure that you will want to access are:

```
sfinfo.samplerate;
sfinfo.channels;
sfinfo.frames;
```

After each call to `sf_open()`, print the index of the file in `ifile_list.txt` and the WAV file information from the corresponding `SF_INFO` header: number of frames, number of channels and sampling rate and WAV filename as shown here.

```
0 Frames:   240000, Channels: 2, Samplerate: 48000, ./signals/sig1.wav
1 Frames:   240000, Channels: 2, Samplerate: 48000, ./signals/sig2.wav
(etc.)
```

Check that each WAV file

- Has a number of channels no greater than MAX_CHAN (which is a #define)
- Has the same sampling rate.
- Has the same number of channels.

And write to the callback `buf` (common for all files)

```
p->channels = sfinfo.channels
p->samplerate = sfinfo.samplerate;
```
One way to check that all files have the same sampling rate and number of channels is to write `channels` and `samplerate` to the callback `buf` for the first file (`i == 0`) and for each subsequent interation through the `for()` loop, check that
```
p->channels == sfinfo.channels
p->samplerate == sfinfo.samplerate;
```
If not, then print an error and return -1;

Allocate storage for the audio data using `malloc()`
```
p->x[i] = (float *)malloc(
      sfinfo.frames*sfinfo.channels*sizeof(float)) )
```
where `x[i]` is an array of pointers to float (the audio data). If the returned pointer stored in `x[i]` is NULL, report an error and return -1; Each audio frame is typically stereo (channels is 2) and each audio sample will be stored as a float. You don't need to use `calloc()`, which zeros the buffer, since the next step is to load the entire buffer with data.

Read the WAV file data using
```
count = sf_readf_float( sndfile, p->x[i], sfinfo.frames );
```
Use `sf_readf_float()` since this reads frames of audio and converts them into floats. Check that `count` (the number of fames read) equals `sfinfo.frames` (the number of frames in the file). If not, report an error and return -1.

Your program will use *pointer arithmetic* in the callback (more on that below in the discussion on the callback). This will require that the data structure have

- A pointer to the first sample in the WAV data buffer
- A pointer to the last sample in the WAV data buffer (actually, one past the last sample)

Close the WAV file using
```
sf_close(sndfile);
```

(30 points) In callback, play audio data to audio output

Use the PortAudio library callback function to enable playing buffers of audio data to D/A.

In Callback

Fill `output` buffer with audio data associated with selected WAV file. Your program should use pointers instead of array indexes to read and write the audio data.

You are provided with code that initializes a pointer to the portaudio output data buffer:
```
float *po = (float *)outputBuffer;
```

In the body of the callback,

- Initialize a variable `samplesPerBuffer` that is equal to `framesPerBuffer * p->channels`.
- Read the `selection` variable into a local variable. In this way you do the atomic read of `p->selection` only once.
- If `selection` is -1 (`if`), fill the output buffer with zeros. Use a for() loop with limit `samplesPerBuffer.` The loop body is simply:
  ```
  *po++ = 0.0;
  ```
- Otherwise (`else`),
  - Initialize a pointer to the next sample in the buffer
  - Initialize a pointer to the last sample (actually one past the last) in the buffer
  - Use a for loop with loop limit `samplesPerBuffer.` As a first statement in the loop, check if the pointer to the next sample is greater than or equal to the pointer to the last sample in the buffer (using an `if()` statement). If so, then reset to the first sample in the buffer.
  - After this `if()` statement, the loop body should user pointers to copy from the audio buffer to the output buffer, or simply:
    ```
    *po++ = *pb++;
    ```
- After the `for()` loop and as the last statement in this `else` portion, save the value of the buffer pointer (`pb`) to the data structure:
  `p->next_sample[selection] = pb.`