Problem Set 8: Synthesizer

Please send back to me via NYU Classes

- A zip archive named as
  `PS08_<your name as Last_First>.zip`
  containing the C code files that implements all aspects of all problems.

Total points: 100

Points are awarded as follows:

- **20 Points** – Initialize data structure and start portAudio
- **30 Points** – Create Ncurses loop
- **30 Points** – Complete PortAudio callback code
- **20 Points** – Clear code, sensible formatting, good comments

References
The following URL provides a good reference for C language library function usage:
http://www.cplusplus.com/reference/cstdlib/

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

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

Problem Overview

In this assignment you will create a simple 3-octave keyboard synthesizer, where the synthesizer keyboard is the computer keyboard. Each key is associated with a note to play, and each note is the fundamental frequency of that note (that is, there are no harmonics). Only one tone is played at a time.

The tone samples are computed as

```
    v = sin(phase);
```

For each new sample of an output channel, the phase is incremented according to the frequency of the tone. The increment for a frequency f0 is

```
    phase_inc = 2*PI*f0/samping_rate;
```

If f0 is 1 Hz, then after sampling_rate audio samples, the phase will have run from 0 to 2*PI, or one complete cycle. If f0 is 2 Hz, then 2 complete cycles.

Each problem in the assignment creates a portion of the synthesizer or adds additional functionality to the synthesizer. You need to add code where you see the comment

```
    //Your code here
```

You are given the complete code of the following files:

```
synth.h
freq.h
support.c
support.h
build.h
paUtils.c
paUtils.h
```

You are given portions of the code for the main file:

```
synth.c
```

Note: there are no command line arguments for this program, so no need to have a Usage print statement.

(25 Points) Initialize variables, data structure, key2index array and portAudio

Initialize `samp_rate` and `num_chan` from `#define` values.
Initalize `num_chan` in the portaudio callback data structure and the parameters in its `Tone` struct. The Tone struct initial values are:

```
pt->key = -1;
pt->phase_inc = -1;
pt->phase = 0.0;
pt->attack_factor = ATTACK_FACTOR;
pt->decay_factor = DECAY_FACTOR;
pt->attack_amp = 0.0;
pt->decay_amp = 1.0;
```

Copy the portAudio startup and shutdown code to the appropriate locations in the `synth.c` file. (Get the code from the previous homework assignment PS05_WavIO).

(25 Points) NCurses loop

The NCurses initialization consists of this code:

```
initscr(); /* Start curses mode */
cbreak();  /* Line buffering disabled */
noecho(); /* don't echo characters when typing */
```

Then make your program print the following to the NCurses screen buffer using the NCurses function `printw()`:

```
Welcome to my synthesizer!
Keyboard to piano key mapping is:
qwertyQWERTY -> C3 to B3
asdfghASDFGH -> C4 to B4
zxcvbnZXCVBN -> C5 to B5
SpaceBar to quit

Key:
```

Print a space but NOT a CR ('\n') after "Key:" These are lines 0 to 7 in the NCurses screen buffer (line 6 is blank for now). Follow the initial printing with

```
	refresh()
```
To show the NCurses screen buffer in the terminal window.

The NCurses loop is
```
	ch = 0;
	while (1) {
		ch = getch(); // wait for next keypress

		/* process key press */

		/* print information to NCurses screen buffer */

		refresh(); //display NCurses screen buffer
	}
```

NCurses processing should consist of
- Quit if ' ' (spacebar)
- If valid synthesizer key (i.e. key2index[] is not -1) then use the index to lookup the corresponding frequency in the freq[] table:
  ```
  freq_idx = key2index[(int)ch];
  f0 = freq[freq_idx];
  ```
- Use f0 to compute phase increment
  ```
  pt->phase_inc = 2*PI*f0/samp_rate;
  ```
  and (since this is a new frequency), re-initialize:
  ```
  pt->phase = 0.0;
  pt->attack_amp = 0.0;
  pt->decay_amp = 1.0;
  ```
- Then print `freq_idx` on line 7:
  `“Key %3d,     New key: “, freq_idx`
  Otherwise print on line 7
  `“Invalid key. New key: ”`

Note that all printing on line 7 should use NCurses function
`mvprintw(7, 0, "%s", msg);`
Where “msg” is one of the messages above. All of the line 7 messages have the “`Key:` “ characters aligned.

(30 Points) PortAudio callback

Simplest Callback

- The callback will have a loop that runs over the count of output frames, where a frame is the interleaved samples of all channels.
- Within this loop, compute the value for the tone to play out, using the expression
  ```
  v += FS_AMPL * sin(pt->phase);
  ```
  where

  v is the output sample value

pt is a pointer to the Tone array

After v Is computed, the phase in incremented by the phase_inc for that tone.

- Within this loop is a second loop that writes the output v to all channels in the current frame.

Add Attack and Decay

You now have a complete synthesizer, but with two negative features. The first is that one a key is pressed, that tone plays forever, or until a next key is pressed or you exit the program. The second is that the tones start and stop playing as full-scale values. That is not very realistic, and furthermore, causes an audible "pop" when play starts and stops.

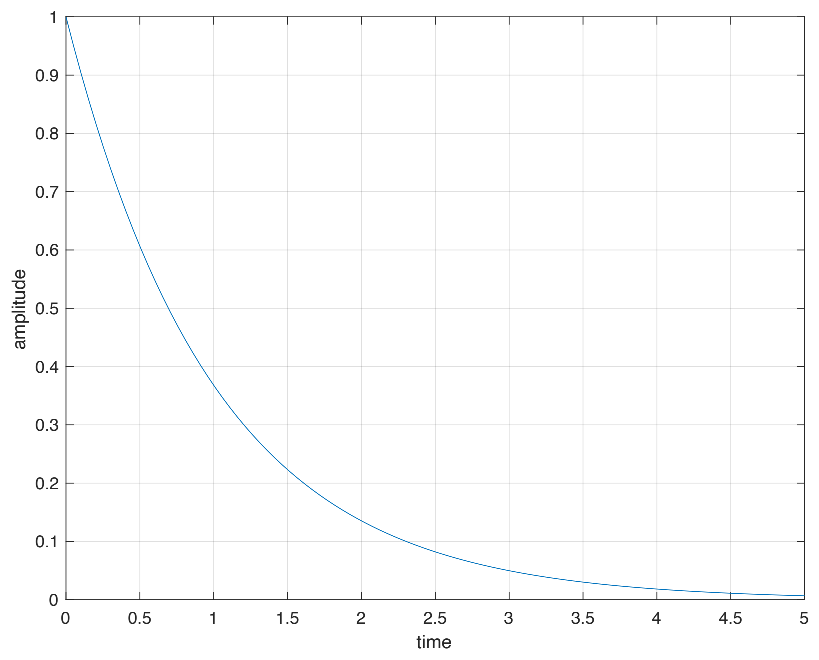
To remove the audible pop and make the synthesizer more realistic, add an "attack" and "decay" to the envelope of the synthesized tone. Since the envelope of a tone now has an exponential decay, you can additionally can stop any tone whose amplitude is below a specified level.

Exponential Decay

An exponential decay is specified by

$$f(t) = e^{-\left(\frac{t}{\tau}\right)}$$

Where τ is the *time constant* and f(t) has the value 1.0 when t has the value 0, and the value 1/e = 1/(2.71828) = 0.368 when t has the value τ . A plot of f(t) for a time constant of 1.0 seconds looks like this:



This can be expressed as a function of n, the digital sample index as

$$f(n) = e^{-\left(\frac{n/Fs}{\tau}\right)}$$

Where Fs is the sampling frequency so that n/Fs is the sample-domain equivalent of t. This can be expressed as

$$f(n) = \left(e^{-\left(\frac{1/Fs}{\tau}\right)}\right)^{n} = (decay\_factor)^{n}$$

Hence, implementing an exponential decay consists of multiplying the output sample value by

$$(decay\_factor)^{n}$$

Where `n` is the sample frame index. This can be computed iteratively. Let

$$decay\_amp(n) = (decay\_factor)^{n}$$

Then

$$decay\_amp(n) = decay\_amp(n-1) * decay\_factor$$

where

$$decay\_amp(0) = 1.0$$

And the indexes are the sample index.

In our program, there is actually just `decay_amp` and `decay_factor`, where `decay_amp` is initialzed to 1.0 when a new key is pressed and, for each sample frame in the output

- the output value is multiplied by `decay_amp and`
- `decay_amp` updated as `decay_amp *= decay_factor`.
- The resultant `decay_amp` value is saved (in the Tone structure).

The value for `decay_factor` corresponding to a time constant of 1.0 seconds is given in the `synth.h` header file.

Exponential Attack

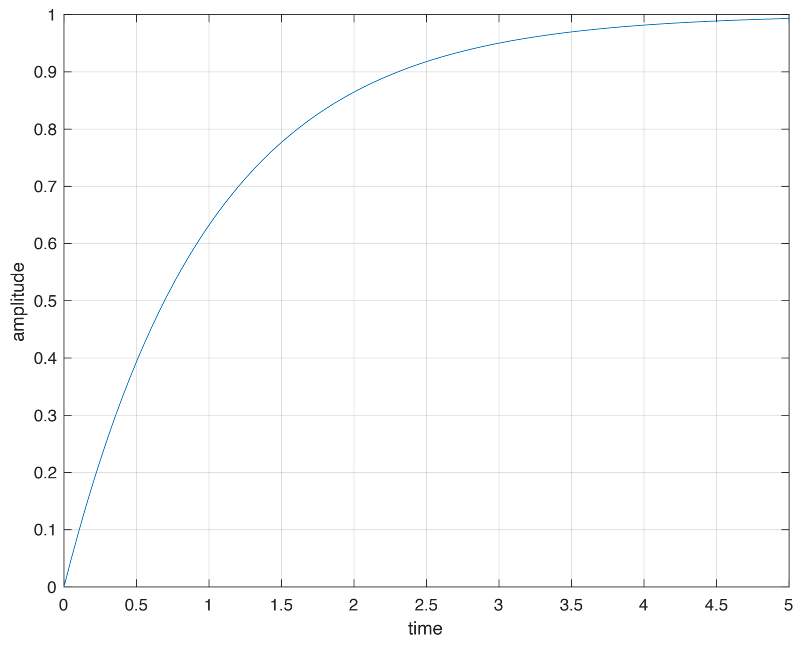
In a similar way, we can add an exponential attack to the tone amplitude

- An `attack_amp` value is initialized to 1.0 when a key is added to the key list (see `add_key(),` above)

However there is one important difference:

- the output value is multiplied by (`1 – attack_amp`)

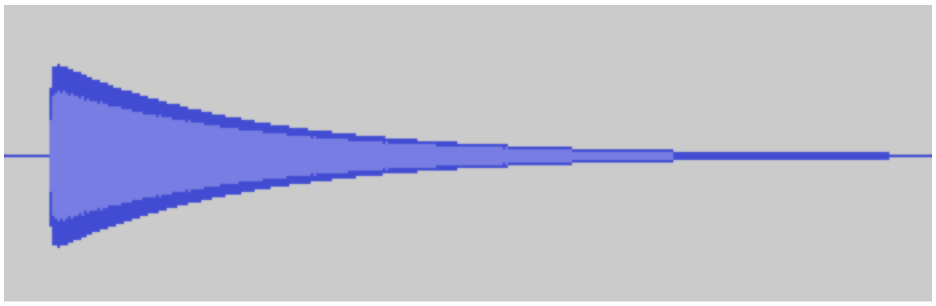
This is a plot of (`1 – attack_amp`) for a time constant of 1.0 seconds:



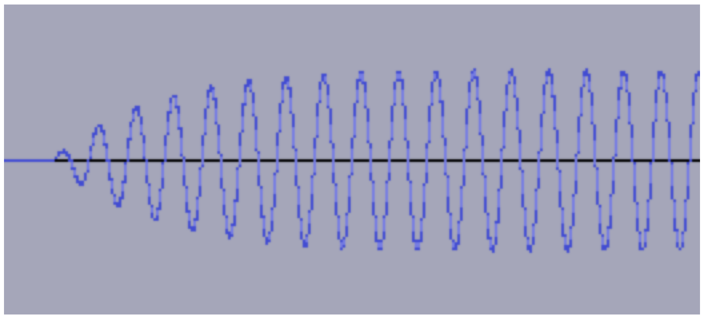
However, a realist value for the attack time constant of e.g. a piano key is perhaps 10 ms, and a realistic decay time constant is 1.0 seconds.

The value for `attack_factor` corresponding to a time constant of 10 ms is given in the `synth.h` header file.

Finally, to apply both attack and decay time constants, simply multiply the output value by `(1 – attack_amp)` and `(decay_amp)`. At the key onset, the attack will dominate the decay, but will quickly reach a value near 1.0. As the tone rings out, the decay will dominate. This will produce a tone with an envelope like this:



The exponential decay is clearly visible. The horizontal time scale for the plot is approximately 5 seconds. Notice that the tone drops to zero amplitude at one point. This is because the program drops the tone from the key list when the envelope falls below the `DROP_LEVEL`. Zooming in on the tone onset shows the attack, where the horizontal time scale is approximately 50 ms.



Task Summary

So, you must add code that does the following:

- Prior to the loop in the callback, add code that stops the tone playout if the amplitude of that tone is below some `DROP_LEVEL` (set to -40 dB in the `synth.h` header file). Stop the tone by setting
  `pt->phase = -1;`
- Inside the loop in the callback, multiply the output sample value by
  `(1 – attack_amp) and (decay_amp)`, and then update the `attack_amp` and `decay_amp` factors.