Problem Set 5 -- PortAudio

Total points: 80
Use the structure in file tone_portaudio.c and fill in code under the comment blocks Problem 1, 2 and 3.

Install PortAudio
On PC/Windows 7
Run Cygwin setup program, search for "portaudio" and install
If you don't have it already, search for "make" and install ONLY the "devel" selection.

On Mac OS X
Open Terminal window and "cd" to the directory with ps5 files.
Execute

cp portaudio.h /usr/local/include
cp libportaudio.dylib /usr/local/lib

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

Problem 1: Command Line Parsing (30 points)

Create a program that has the following command line usage:

```
main [–f freq_in_Hz] [–a level_in_dBFS] [-s sampling_freq_in_Hz]
```

where [ ] indicates optional arguments.

Set default values
Initially, set to:

```
double f0= 440; // frequency of tone to play out
double fs = 48000; // sampling frequency of play out
int level_dBFS = -24; // level of play out, dBFS
```

Parse command line
Parse the command line and overwrite the values as appropriate. If parsing fails, print an error diagnostic and exit.

Consider testing

```
if (argv[i][0] == '-')
```

to determine if –f or –a or –s and their associated arg are present on command line. If they are present, then either as

```
switch (argv[i][1]) {
case 'f':
    …
```

```
            break;
        …
        }
```
statement or a set of if else if statements as
```
        if (argv[i][1] == 'f') {
        …
        else if () {
        …
```
to test if `argv[i][1]` is equal to 'a', 'f' or 's'. Consider using atoi() to convert an arg string to a integer value. This requires #include <stdlib.h>

Print parameter values
Print the values of f0, fs, level_dBFS.

Problem 2: Tone generation (10 points)

Define a structure
```
        typedef struct sound_data {
            float *tone;
            unsigned int tone_len;
            unsigned int next_samp;
            unsigned int count;
        } SoundData;
```

And declare an instance
```
        soundData data;
```

Set tone parameters
```
        tone_len = fs;
        next_samp = 0;
        count = 0;
```

Convert `level_dBFS` to amplitude as:
```
        double level = pow(level_dBFS/20.0);
```
This requires #include <math.h>.

Allocate storage for tone.
Using malloc(), allocate to *tone sufficient storage for tone_len samples. Do this with error checking.

Initialize tone
Fill the tone[] array with a sine wave signal.
```
        tone[i] = level*sin(2*PI*i*f0/fs);
```

Problem 3: Play tone to audio output (40 points)

Use the PortAudio library to enable playing the tone using a callback function. See makefile for how to compile and link with PortAudio

In Main program

This is already done in the example code. Use the structure in file tone_portaudio.c to play a tone to the laptop built-in speaker.

In PortAudio, the program contains a loop after playout starts:

```
while (data.count < fs * 5) {
    sleep(1);
}
```

This monitors data.count and exits after 5 seconds of tone data.

In Callback

Fill buffer from tone data

In the body of the callback, copy framesPerBuffer floats (or BUFFER_SIZE / sizeof(SAMPLE_TYPE), which is the same thing) from tone[]to the callback buffer out[] (or casted_buffer[]). The first sample to copy is at index next_samp. If a next sample would be beyond the end of the tone[] array, wrap back to the start of tone[] by setting the index to zero. In addition, increment count for each sample. After the copy, set next_samp to the next sample needed for the next callback. For example:

```
K = pt->next_samp;
for (i=0; i<N; i+=NUM_CHN) {
    if (k >= pt->tone_len) {
        k=0;
    }
    for (j=0; j<NUM_CHN; j++) {
        buf[i+j] = pt->tone[k];
        buf[i+j] = pt->tone[k];
    }
    j++;
}
pt->next_samp = k;
```