Problem Set 5: Monte Carlo Methods, Reading WAV Files

Please send back via NYU Classes
- A zip archive named as
  `PS05_<your name as FirstLast>.zip`

containing the C code files that implement all aspects of all problems.

Total points: 100

Problem 1 (20 points): Monte Carlo simulation to find the value of *e*

`montecarlo_e.c`
Similar to the Monte Carlo method to find the approximate value of π, there is a simple method to find the approximate value of *e*, the base of the natural logarithm.

In this method, you repeatedly draw a random real number uniformly distributed between 0 and 1, and sum these numbers until the sum becomes larger than 1. The estimation of the Euler's number *e* is equal to the count of how many random numbers you draw *before* the sum exceeds 1.0. If you repeat this experiment many times, the average of the counts converges to *e*.

Write a function that implements a single iteration of this approximation method, and a program that calls the function ten million times, and prints the resulting approximate value of *e*. Is it similar to the real value? The value of *e* is 2.7182818284...

Note: You can sample a random number between 0 and 1 by using rand() / (float)RAND_MAX. You must use #include <stdlib.h> in order to use rand().

Problem 2 (80 points): Read and write WAV file

`process_wav.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 program should have the following command line usage:

```
./process_wav –max amplitude ifile ofile
```

`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

If the correct arguments are not present, the program should print a usage statement. Consider using **strncmp()** to determine if –max is present on command line. This requires #include <string.h>. Consider using atoi() to convert an arg string to a integer value. This requires #include <stdlib.h>.

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 might also want to review **Big-Endian** and **Little-Endian**.

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

```
    if ( (ifp = fopen(ifile, “r”)) == NULL) {
                // error 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) != 1 ) {
                // error reporting
    }
```

Print each byte of the wav header as character and hexadecimal values as 4 lines of 11 characters per line. If `header[i]` is an ASCII character, then print the character, otherwise print a SPACE character. You might use this code to print each character:

```
    c = isalpha(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. Print the following information about the WAV file:

- Number of channels
- Sampling Rate
- Bits per sample
- The data size, in bytes, of the sound data portion of the WAV file.

Note that header[] is a unsigned char array and the values above are 16 or 32 bit integer values. Also, note that they are Little-Endian in their byte layout. You can construct the sampling rate as:

```
fsamp = header[27]<<24 | header[26]<<16 | header[25]<<8 | header[24];
```

Confirm that the data size is consistent with the file 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;
      if ( (x = (short *)malloc(N * sizeof(*x)) == NULL) {
            // error reporting
      }
```

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

```
      if ( fread(x, sizeof(*x), N,), 1, ifp) != 1 ) {
            // error reporting
      }
```

The maximum absolute value can be calculated as follows:

```
      int vmax = 0;
      for (i=0; i<N; i++) {
            if (abs(x[i]) > vmax) {
                  vmax = 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

$$xnew = (x)\left(\frac{target\_value}{x\_value}\right)$$

where *target_value* is the value given on the command line and *x_value* is the calculated max abs value method given above.

You can overwrite the x array with the adjusted values (*xnew*, above). 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. 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.