CSC 101 - Fall 2012
Lab 7 - Digital Audio

1. Digital audio is represented by a sequence of numbers. There are two key considerations which affect sound fidelity : **sample rate** and **discretization error**. The **sample rate** refers to how many numbers are used to represent one second of sound, and is given in terms of *samples per second*. The **discretization error** depends on how many bits are used to represent each number.

   (a) For CD-quality audio, what is the sample rate ?

   (b) For CD-quality audio, what is the number of bits used for each sample ?

2. Digital audio file formats such as MP3, Au, Wav, Ogg, and many others allow some type of **data compression** to reduce file size. An in-depth study of the compression techniques used in these file formats is beyond the scope of our course. However, we can get the basic idea: data compression refers to any technique which allows information to be represented in less memory than would be used by storing all data items using their usual representation. For this exercise, we suppose we have data represented in a six letter alphabet with the probabilities (frequency of occurrence) shown below:

Letter	**Probability**
A	0.40
B	0.10
C	0.22
D	0.05
E	0.15
F	0.08

   (a) Draw a Huffman tree for the table above. For clarity, draw your tree with the leaves (letters) in the order: A, C, E, B, D, F

   (b) Give a Huffman code corresponding to your Huffman tree.

   (c) A six letter alphabet requires three bits per letter. Compute the expected value (average number) of the number of bits needed for each letter using your code.

For the next part of our lab we will use a software called **Matlab**. **Matlab** is an interactive software designed primarily for numerical computation with matrices and vectors. Its vector handling features will greatly simplify working with sound samples. To get started, first make sure you are connected to the WFU network. Go to the windows logo in the lower left corner of your screen and use it to find "Matlab". Click on **MATLAB R2012a**. Be patient. Matlab takes a while to load. Matlab should display a partitioned window; we will work primarily by writing Matlab language statements in the center window.

The Matlab language has powerful tools for working with arrays of numbers, so it will not be necessary for us to write loops.

3. Our first task is to just play a sound to make sure everything is working correctly. Matlab has some sound samples built-in so we will use one of those. In the Matlab window, type:

```
load handel;
p = audioplayer(y, Fs);
play(p);
```

The **load** statement reads a Matlab file. More than one variable may be stored in a Matlab file ; in this case two variables are provided. The variable **y** is an array of sound samples. The variable **Fs** is the sample rate. What is the sample rate for this sound? *Hint:* To see the value of a variable in Matlab, simply type the name of the variable (without the semicolon) and press the enter key. I.e.,

```
Fs
```

The statement `p = audioplayer(y, Fs);` converts the sample numbers from `y` into Matlab's preferred representation of a sound object. The statement `play(p);` plays the sound object named "p".

4. Our second task is to look at a graph of the sound signal. We can look at a small part of the sound sample array by using Matlab's clever array indexing features. Type:

```
plot( y( 1:128 ) ) ;
```

Some comments about the Matlab language: the notation `1:128` generates an array of numbers $[1, 2, 3, 4, ..., 128]$. Array indexing is done using parentheses, i.e., ( ). You will recall a similar concept in the C language, except in C we used square brackets, i.e., [ ]. Unlike C, Matlab allows the index to be an array of integer positions. The notation `y(1:128)` creates an array of the first 128 numbers from the array `y`. Also unlike C, arrays start at position 1 in Matlab, not in position 0.

Print the plot (to a file) using the menus on the plot window. Use a the file name "lab7_plot" to print the plot. Upload your plot to your account on menehune as part of turning in this lab.

5. Our third task is to look at a file containing the numbers representing a sampled sound, and manipulate that sound. For this task, point your browser at:

```
http://menehune.opt.wfu.edu/csc101/Sound
```

You should see a file called `snums`. Download that file and move it to the folder in which your Matlab session is running. To find out the name of this folder, go to the Matlab window and type:

```
pwd
```

You should see `\userdata\Documents\MATLAB`.

What is the first number in the downloaded file ?

If you are having problems downloading via your browser: Alternately, you can use `IPswitch WS_FTP` to copy the file `snums` from your directory `Lab7` on menehune to your laptop computer.

Back in the Matlab window, we are going to load these numbers into Matlab and treat it as a sound file. Type:

```
load snums ;
```

The file named `snums` is an ASCII file, and Matlab will treat it a little differently than the file named `handel`. For the file name `snums` Matlab's load statement will create an array named "snums" containing all of the numbers from the file. We now want to hear the sound.

```
q = audioplayer(snums, 44100);
play(q);
```

6. Ok, now we can play with our sound. Let's hear it backwards. First, we reverse the array. Our array has 551250 numbers in it. We can use Matlab's indexing features to create an index array that counts downward. Type:

   ```
   x = snums( 551250:-1:1 )
   ```

   to reverse the array. Then use the familiar `audioplayer` and `play` functions to play the sound. I.e.,

   ```
   r = audioplayer(x, 44100);
   play(r);
   ```

7. For our next trick, we will synthesize a sound using numerical calculation alone. We want to hear a 5 second sound of a pure sine wave at 440 Hz. We choose a sample rate of 8192 samples per second. Here is how to do that in Matlab: **NOTE: Do not forget the semicolon at the end of the statement !!**

   ```
   t = (1:(5*8192)) ./ 8192 ;
   z = sin( 2.0 * pi * 440.0 * t );
   ```

   The statement `t = (1:(5*8192)) ./ 8192 ;` creates an array of time values with an inter-sample time period of 1/8292 seconds. The statement `z = sin( 2.0 * pi * 440.0 * t );` creates an array of sample values. We then play it with our familiar functions, audioplayer, and play.

   ```
   s = audioplayer(z, 8192);
   play(s);
   ```

8. Ok. Flushed with success, we synthesize a higher pitched tone.

```
z2 = sin( 2.0 * pi * 440.0 * ( 5.0 / 4.0 ) * t );
s2 = audioplayer(z2, 8192);
play(s2);
```

Optional Challenge for the musically knowledgeable: The tone represented by the array `z` is 440 Hz. Musically, that is A above middle C. Here's the challenge: What is the note represented by `z2` ?

Optional Super Challenge for the musically knowledgeable: is the note represented by `z2` in "equally tempered intonation", or in "true intonation" ?

9. Ok. Now we are on a roll. Let's hear both sounds at once.

```
temp =  z + z2 ;
z3 = temp ./ max(temp) ;
s3 = audioplayer( z3 , 8192) ;
play(s3);
```

10. For our final audio trick we will create an echo effect. The echo effect is most easily heard with our sound from the file `snums`. The sampling rate is 44100, and we would like our echo effect to separate the echo by 1/6 of a second. Ok, 44100 / 6 = 7350 samples. We can delay a sound in time by padding the beginning of the array with zeros. To do this in Matlab:

```
u1 = [ zeros( 7350, 1) ; snums ] ;
u2 = [ snums ; zeros( 7350, 1) ] ;
u3 = u1 + u2 ;
u4 = u3 ./ max(u3) ;
s4 = audioplayer( u4 , 44100 ) ;
play(s4);
# comment:  compare sound object s4 to sound object q
play(q);
# comment:  Do you hear the echo effect ?
```