CSC 101 - Spring 2013
Lab 8 - Digital Audio ANSWERS

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 ?

   Answer: 44100 samples per second

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

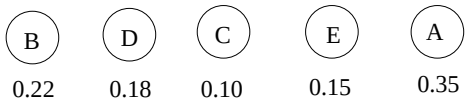
   Answer: 16 bits (2 bytes) per sample

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 five letter alphabet with the probabilities (frequency of occurrence) shown below:

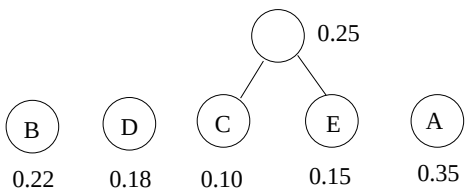
Letter	Probability
A	0.35
B	0.22
C	0.10
D	0.18
E	0.15

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

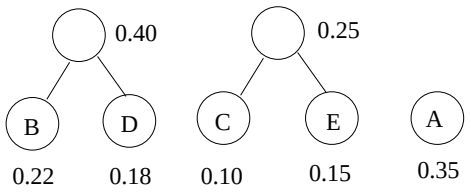
   Answer: We start with each letter as a single node:



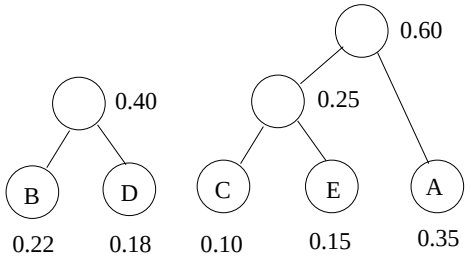
   We then select two nodes with the lowest probabilities, create a new node, and link the selected nodes as "child nodes" of the new node. We then label the new node with the sum of the probabilities of the two child nodes.



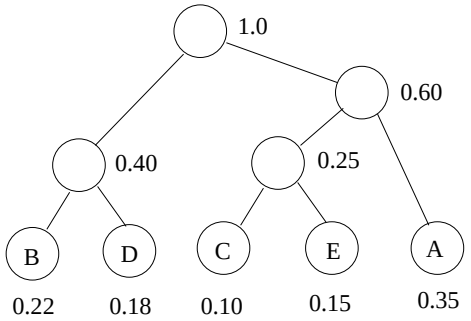
We repeat this process. The next two nodes with the lowest probabilities lead to the following (partial) tree.



We only select from the nodes which have no ancestors. In the situation above, we select the nodes labeled "0.25" and "0.35".



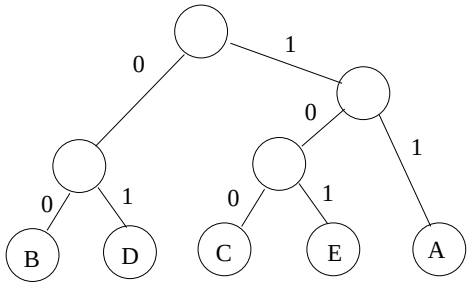
Finally, we have:



And, our Huffman tree is done.

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

Answer: We label each left branch with a "0" and each right branch with a "1". The code is then formed by the labels along the path from the top node to the letters at the bottom of the diagram.



The table below gives the codes:

A	B	C	D	E
11	00	100	01	101

(c) Is any code in your encoding scheme the prefix of any other code ?

Answer: No.

(d) A five letter alphabet requires three bits per letter to create a fixed-length code. Compute the expected value (average number) of the number of bits needed for each letter using your code. Is it better than a fixed length code ?

Answer:

$$2 \times 0.35 \;+\; 2 \times 0.22 \;+\; 3 \times 0.10 \;+\; 2 \times 0.18 \;+\; 3 \times 0.15 \;=\; 2.25$$

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
```

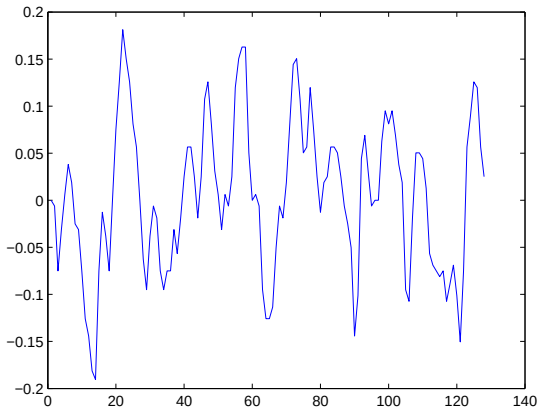
The statement `p = audioplayer(y, Fs);` converts the sample numbers from `y` into Matlab's preferred representation of a sound object. The statement `play(p);` plays the sound object named "p".

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., `( )`. Matlab allows indexing an array with another array.

Answer:



Print the plot (to a file) using the menus on the plot window. Use a the file name "lab8_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/Lab8`

   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 ?

   Answer: -0.011

   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. We need to reverse the array. To do that, first find out how long the array is using Matlab's `size()` function. Then, use the index generating features (colon notation), and array indexing to create a new array with the sample values in the reverse order.

   Use the familiar `audioplayer()` and `play()` functions to play the sound.

   Answer:

```
>  size( snums )
ans =

    551250        1

>  newsnums = snums( 551250 : -1 : 1 ) ;
>  pnew = audioplayer( newsnums, 44100 ) ;
>  play( pnew ) ;
```

7. For our next trick, we will synthesize a sound using numerical calculation alone. We want to hear a two second sound of a pure sine wave at 440 Hz. This is the standard frequency for the note "A" above middle-C. We choose a sample rate of 8192 samples per second.

   (a) Generate a time vector with sample rate of 8192 samples per second spanning time values from 0 to 2. NOTE: Do not forget the semicolon at the end of the statement !

   (b) For frequency f our formula for the sample values $s(t)$ of a pure sine wave sound is:

$$s(t) = \sin(2\pi f t)$$

   You will need to give a variable name to the array that holds the sample values. Use the name `sa`. Play this sound with our familiar functions, `audioplayer()`, and `play()`.

   Answer:

```
>  t = 0 : ( 1/8192 ) : 2 ;
>  sa = sin( 2.0 .* pi .* 440.0 .* t ) ;
>  psa = audioplayer( sa, 8192 ) ;
>  play( psa ) ;
```

8. Ok. Flushed with success, we synthesize a higher pitched tone. The frequency of note C♯ is $5/4 \times 440$ in "true intonation" tuning and $2^{1/3} \times 440$ in "equal temperament" tuning. If you are not a violin player, use equal temperament tuning. You will need to give a variable name to the array that holds the sample values. Use the name `scs`.

   Answer:

```
> scs = sin( 2.0 .* pi .* 2^(1.0/3.0) .* 440.0 .* t ) ;
> pscs = audioplayer( scs, 8192 ) ;
> play( pscs ) ;
```

9. Ok. Now we are on a roll. Let's hear both sounds at once. Add the sounds `sa + scs`, and play the added sounds. Store the new sound in an array named `both`.

   Answer:

```
> both = sa + scs ;
> pboth = audioplayer( both, 8192 ) ;
> play( pboth ) ;
```

10. Let's hear both sounds in sequence. Use the vector concatenation feature of Matlab to create a sequence A, C♯, A, C♯, A. Use the variables `sa` and `scs` you have created previously. Store the new sound in an array named `sascs`.

    Answer:

```
> sascs = [ sa , scs , sa , scs ] ;
> psascs = audioplayer( sascs, 8192 ) ;
> play( psascs ) ;
```

11. 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 ] ;
```

    When adding arrays in Matlab, they must be the same length. Create another array named `u2` that has the zeros padded at the end of the array. Create a new array `u3 = u1  u2 ;+`. Play both `u1` and `u3`. Do you hear the echo effect.

    Answer:

    To concatentate column arrays we need to separate them using ";" instead of a comma.

```
> u1 = [ zeros( 7350, 1) ; snums ] ;
> u2 = [ snums ; zeros( 7350, 1) ] ;
> u3 =  u1 + u2 ;
> pu3 = audioplayer( u3, 44100 ) ;
> play( pu3 ) ;
```

12. Another way to create an echo effect is with a convolution. Create an array named `irf` of all zeros of length 22050. Put a 1 in positions 1, 7350, 14700, 22050 in the array. Then compute:

```
uu = conv( snums, irf ) ;
```

Note: This may take some time.

Finally, play the samples you have stored in the array `uu`.

Answer:

```
>  irf = zeros( 22050, 1) ;
>  irf(1) = 1 ;
>  irf(7350) = 1 ;
>  irf(14700) = 1 ;
>  irf(22050) = 1 ;
>  uu = conv( snums, irf ) ;
>  puu = audioplayer( uu, 44100 ) ;
>  play( puu ) ;
```