Lab 3

This lab requires you to read an abc file. I did this lab by using three vectors. I did not use a stack for repeat because my conversion process handles the repeat process as well. I did not use stack in it so I did not test Catch 2.

lab.h

I tried this lab at home so I only used a "Sound" structure. The key difference is that I read the abc file by char instead of string. The advantage to using char is no need to hard edit the abc files, and it is easier to manipulate the values of the "note" of each sound.

```
struct Sound
{
    char note;
    double duration;
    int octave;
};
```

The functions I used include readABCfile (taken from class), converter (converts abcsound into actual notes), measureconverter (converts notes into readable music using tempo from gettempo), and playSong (plays each note in a for loop). It can be noted that converter and measureconverter both take in a vector and outputs a vector. In other words, from raw abc characters to music the system can understand is accomplished by two conversions.

```
bool readABCfile(string, vector<Sound> &);
bool converter(vector<Sound> &insound,vector<Sound> &outsound);
bool measureconverter(vector<Sound> &insound,vector<Sound> &outsound, double &tempo);
double gettempo(vector<Sound> &insound,double &tempo);
bool playSong(vector<Sound> &insound,double &tempo, double userspeed);
```

Convert ABC into Musical Notes

In this function, the abc characters are converted into characters with with three attributes: note, duration, and the octave of the note. Only meaningful notes are let into the party, that is why I had to hard code 'a' to 'g' and 'A' to 'G'. "temp" is a temporary Sound struct that holds the information about the note and adds it to the output vector.

```
else if(currentnote== "a" || currentnote== "b"|| currentnote== "c" ||
currentnote== "d" ||currentnote== "e" ||currentnote== "f" ||currentnote== "g")
{
    temp.note=insound[i].note;
    temp.duration=1;
    temp.octave =5;
    outsound.push_back(temp);
    counter++;
}
else if (currentnote== "A" || currentnote== "B" || currentnote== "C" ||
currentnote== "D" ||currentnote== "E" ||currentnote== "F" ||currentnote== "G" )
{
    temp.note=insound[i].note;
    temp.duration=1;
    temp.octave =4;
    outsound.push_back(temp);
    counter++;
}
```

The notes are the VIPs of the converter because they're actual "notes" in real life. So they're given immediate entry into the output vector. The converter then also understands numbers and '/' which provide useful information to the real note.

Convert ABC into Musical Notes

The true agents that run the operation and provide information for the next converter are the ':' and vertical bar characters. So these guys are also allowed into the output vector.

```
else if(insound[i].note == ':')
{
    if(insound[i-1].note == 'K')
    {mcounter++;}
    else
    {
        if(mcounter>0)
        {   temp.note=insound[i].note;
            temp.duration=0;
            temp.octave=0;
            outsound.push_back(temp);
            counter++;}
    }
}
```

Convert ABC into Musical Notes

The vertical bar character and ':' both help rearrage the notes for the next conversion.

```
else
{
    if(insound[i].note == '|')
    {
        if(insound[i-2].note == ':')
        {if(insound[i-3].note == 'K')
            insound[i].note = 'n';}
        else
        {
        temp.note=insound[i].note;
        temp.duration=0;
        temp.octave=0;
        outsound.push_back(temp);
        counter++;
        }
    }
```

Convert ABC into Musical Notes

Here is my code for the number handling process:

```
Sound temp, repeattemp;
string currentnote,pastnote;
int counter=0, mcounter=0;
for(int i = 0; i < insound.size();i++)
{
    currentnote = insound[i].note;
    pastnote = insound[i-1].note;
    if(insound[i].note == '2' && mcounter >=1)
    {
        if(insound[i-1].note== '/')
        {outsound[counter-1].duration=0.5;}
        else
        {outsound[counter-1].duration=2;}
    }
     if(insound[i].note == '3' && mcounter >=1)
    {
        if(insound[i-1].note== '/')
        {outsound[counter-1].duration=1/3;}
        else
        {outsound[counter-1].duration=3;}
    }
    else if (insound[i].note == '4' && mcounter>=1)
    {
        if(insound[i-1].note== '/')
        {outsound[counter-1].duration=0.25;}
```

```
    else
    {outsound[counter-1].duration=4;}
}
else if (insound[i].note == '8' && mcounter>=1)
{
    if(insound[i-1].note== '/')
    {outsound[counter-1].duration=0.125;}
    else
    outsound[counter-1].duration=8;

}

else if (insound[i].note == '6'&& mcounter>=1)
{
    if(mcounter>=1 && insound[i-1].note == '1')
    {
        if(insound[i-2].note== '/')
        {outsound[counter-1].duration=.0625;}
    }
}
```

```
 else if (currentnote =="'")
{
    if(pastnote=="'")
    {
        if(outsound[counter-1].octave == 6)
            outsound[counter-1].octave++;
    }
    else
    {
        if(outsound[counter-1].octave ==5)
            outsound[counter-1].octave++;
    }
}
 else if (currentnote ==",")
{if(outsound[counter-1].octave ==4)
    outsound[counter-1].octave--;}
```

The numbers should be self-explanatory, if there is a '/' infront of the number such as /4, then that means the duration is divided by four instead of multiplied. The ' ' ' and ' , ' characters are attributes to the note. If these are referenced, then the note that already entered the output vector's octave would be "corrected".

Get the Tempo

This function grabs the char tempo from abc sound and directly converts the character to its double value. From this I can just use pass by reference to let it go between functions.

```
#include "lab.h"
double gettempo(vector<Sound> &insound,double &tempo)
{
    char charholder; double d1,d2;
    for(int i = 0; i < insound.size()-5;i++)
    {
        if(insound[i].note ==’L’)
        {
            if(insound[i+1].note==’:’)
            {
                if(insound[i+5].note == ’6’)
                    tempo = 16;
                else if(insound[i+5].note == ’2’)
                tempo = 32;
                else
                {
                    charholder = insound[i+4].note;
                    d1 = (double)(charholder);
                    tempo = d1-48;
    }}}}
    return tempo;
```

Convert Musical Notes to System Readable Notes

In measure converter, I create a temporary Sound structure "temp" and also a vector of Sound that will only appear for repeats in the ABC Notation. The purpose of this measure converter is to "grab" correct notes into the stream. That is why vertical bar and ':' are so important. The tempo is also passed in.

```
#include "lab.h"
bool measureconverter(vector<Sound> &insound,vector<Sound> &outsound,double & tempo)
{
    Sound temp;
    vector<Sound> repeattemp;
    int pos, counter=0, repeatsignal=-1, repeatcounter = 0;
    double time;
```

Convert Musical Notes to System Readable Notes

In a for loop, everytime a vertical bar is come across in the Musical Notes, it counts back the amount of time until the tempo is reached. The position is recorded by "pos". the data is filled into temp and then pushed back.

```
for(int i=0; i<insound.size();i++)
{
    if(insound[i].note =='|')
    {
        if(insound[i-1].note != ':')
        {
            time =0;pos=1;
            while(time!=tempo && insound[i-pos].note !='|')
            {
                time=insound[i-pos].duration + time;
                pos++;
            }
            for(int p = pos;p>0;p--)
            {
                temp = insound[i+1-p];
                outsound.push_back(temp);
            }
        }
    }
```

Convert Musical Notes to System Readable Notes

The ':' character is more complicated, since it has to deal with the repeats. I worked around this by using a repeat signal that signifies "ON" or "OFF" when a repeater character ':' is passed.

```
        if(insound[i].note == ':')
        {
            repeatsignal=repeatsignal*-1;
            if(repeatsignal == -1)
            {
                pos = 1; time=0;
                while(time != tempo && insound[i-pos].note !=':')
                {
                    time = time + insound[i-pos].duration;
                    pos++;
                }
                //cout << "Position is: "<<pos<<endl;
                for(int p = 0; p<pos-1;p++)
                {
                    repeattemp.push_back(insound[i-(p+1)]);
                }

                for(int set = 0; set< 2; set++)
                {
                    for(int rev = 0; rev < repeattemp.size() ; rev++)
                    outsound.push_back(repeattemp[repeattemp.size()-rev-1]);
                }
                repeattemp.clear();
            }
    }
```

Vector Conversion Output

Here are what ther three vectors go through:

```
Enter file name to play: frerejacques2.abc
===============Measure conversion complete===============

===============abc sound===============
% G e n e r a t e d m o r e o r l e s s a u t o m a t i c a l l y b y s w t o a b c b
 y E r i c h R i c k h e i t K S C X : 1 T : F r e r e J a c q u e s M : 4 / 4 L : 1
/ 4 % % m e a s u r e n b 1 K : C n c d e c | c d e c | e f g 2 | e f g 2 | g / 2 a /
 2 g / 2 f / 2 e c | g / 2 a / 2 g / 2 f / 2 e c | c G c 2 | c G c 2 | C 4 | |
===============actualnotes===============
G e e a e d e e a a c a b a b c b E c c e C F e e a c e e a e b C c d e c | c d e c |
 e f g | e f g | g a g f e c | g a g f e c | c G c | c G c | C | |

===============forrealnotes===============
c d e c | c d e c | e f g | e f g | g a g f e c | g a g f e c | c G c | c G c | C | |
```

readABCfile

Takes the abc file and inputs the character stream into my abcnotes Sound vector. This was taken from lab in class.

```
#include "lab.h"
bool readABCfile(string filename, vector<Sound> & v )
{
    bool r = true;
    ifstream ifs(filename);
    if(ifs)
    {
        Sound s;
        while(ifs>>s.note)
            v.push_back(s);
    }
    else {cout<<"Error opening file" <<endl; r = false;}
}
```

create Play Cmd

Takes the stream of "forrealnotes" from main and plays the correct note, octave, and duration:

```
#include "lab.h"
bool playSong(vector<Sound> &insound,double &tempo, double userspeed)
{
    string playCad, stringnote, notespeed, chartostring, uppernote, cmd="";
    double notespeedm;

    for(int i=0; i<insound.size();i++)
    {
        if(insound[i].note!='|' && insound[i].note!=':')
        {
            playCad = "play -n synth 1 pluck NOTE0"; //Declare play command
            size_t pos = playCad.find("NOTE0"); //Find the position to replace
            uppernote = toupper(insound[i].note); //upper case
            chartostring = string(uppernote);//turn from char to string
            stringnote = chartostring + to_string(insound[i].octave); //Add octave(double to st
            playCad.replace(pos,5,stringnote); //Replaced
```

create Play Cmd

I added "userspeed" so the user can determine the pace of the song. My favorite userspeed is x1.20

```
            //Replace the speed
            notespeedm=(insound[i].duration * userspeed)/(tempo/2) ;//My own personal tweak to
            notespeed = to_string(notespeedm); //turn double into string
            size_t speedpos = playCad.find("1"); // find position of "1"
            playCad.replace(speedpos,1,notespeed); // replace the 1 with the speed
            //cout << "This note's string is: "<<playCad<<endl;
            system(playCad.c_str());
        }
    }
```

Main

I will display all of Main before I comment on it

```
#include "lab.h"
int main()
{
    //Declare 3 vectors
    vector<Sound> abcsound;
    vector<Sound> actualnotes;
    vector<Sound> forrealnotes;
    double tempo, userspeed;
    int octave; string cmd = "", filename;

    //Prompt user for the abc file to play
    cout << "Enter file name to play: ";
    cin>> filename;

    //Read abc file, get the tempo, convert raw char abc into actual notes, then convert notes
    if(readABCfile(filename,abcsound))
    {
        gettempo(abcsound,tempo);
        if (converter(abcsound,actualnotes))
        {
            if(measureconverter(actualnotes,forrealnotes,tempo)) //Convert raw sound to measure
            {//Notes are now playable by system
                cout << "===============Measure conversion complete==============="<<endl<<endl
        }
    }
```

```
    //Displaying each version
    cout << "===============abc sound==============="<<endl;
    for(int i =0; i<abcsound.size();i++){cout <<abcsound[i].note <<" ";}; cout << endl;
    cout << "===============actualnotes==============="<<endl;
    for(int i =0; i<actualnotes.size();i++){cout <<actualnotes[i].note <<" ";}; cout << endl;
    cout <<endl<< "===============forrealnotes==============="<<endl;
    for(int i =0; i<forrealnotes.size();i++){cout <<forrealnotes[i].note<<" ";} cout << endl<<e


    //Pop open the image before play Song
    //cout << "What speed would you like to play at? (recommended 1.20): "; cin >> userspeed;
    cmd += "abcm2ps -E " + filename + " && display -alpha off Out001.eps &";
    system(cmd.c_str());
    playSong(forrealnotes, tempo, 1.20); //Play the song
    system("pkill -9 display"); //Close the image
    cout << "Song over"<<endl;
    return 0;
}
```

Main

I first converted the raw abc file twice, then displayed each one so I could keep track of each vector, then I open the image before the song is playing . I play the song then close the image after the song is done.

Runtime

Here is what the system sees to play:

```
This note's string is: play -n synth 0.600000 pluck C5
This note's string is: play -n synth 0.600000 pluck D5
This note's string is: play -n synth 0.600000 pluck E5
This note's string is: play -n synth 0.600000 pluck C5
This note's string is: play -n synth 0.600000 pluck C5
This note's string is: play -n synth 0.600000 pluck D5
This note's string is: play -n synth 0.600000 pluck E5
This note's string is: play -n synth 0.600000 pluck C5
This note's string is: play -n synth 0.600000 pluck E5
This note's string is: play -n synth 0.600000 pluck F5
This note's string is: play -n synth 1.200000 pluck G5
This note's string is: play -n synth 0.600000 pluck E5
This note's string is: play -n synth 0.600000 pluck F5
This note's string is: play -n synth 1.200000 pluck G5
This note's string is: play -n synth 0.300000 pluck G5
This note's string is: play -n synth 0.300000 pluck A5
This note's string is: play -n synth 0.300000 pluck G5
This note's string is: play -n synth 0.300000 pluck F5
This note's string is: play -n synth 0.600000 pluck E5
This note's string is: play -n synth 0.600000 pluck C5
This note's string is: play -n synth 0.300000 pluck G5
This note's string is: play -n synth 0.300000 pluck A5
This note's string is: play -n synth 0.300000 pluck G5
This note's string is: play -n synth 0.300000 pluck F5
This note's string is: play -n synth 0.600000 pluck E5
This note's string is: play -n synth 0.600000 pluck C5
This note's string is: play -n synth 0.600000 pluck C5
This note's string is: play -n synth 0.600000 pluck G4
This note's string is: play -n synth 1.200000 pluck C5
This note's string is: play -n synth 0.600000 pluck C5
This note's string is: play -n synth 0.600000 pluck G4
This note's string is: play -n synth 1.200000 pluck C5
This note's string is: play -n synth 2.400000 pluck C4
```

Runtime

Here is what the system (debian) looks like when it plays:

```
In:0.00% 00:00:00.68 [00:00:00.00] Out:28.8k [ -====|====- ] Hd:0.0 Clip:0
Done.

  Encoding: n/a
  Channels: 1 @ 32-bit
Samplerate: 48000Hz
Replaygain: off
  Duration: unknown

In:0.00% 00:00:00.68 [00:00:00.00] Out:28.8k [ -====|====- ] Hd:0.0 Clip:0
Done.

  Encoding: n/a
  Channels: 1 @ 32-bit
Samplerate: 48000Hz
Replaygain: off
  Duration: unknown

In:0.00% 00:00:00.68 [00:00:00.00] Out:28.8k [ -====|====- ] Hd:0.0 Clip:0
Done.

  Encoding: n/a
  Channels: 1 @ 32-bit
Samplerate: 48000Hz
Replaygain: off
  Duration: unknown

In:0.00% 00:00:00.68 [00:00:00.00] Out:28.8k [      |      ] Hd:0.0 Clip:0
```