**CS Project 3 report**

1. **Brief description of obstacles:**
   1. I think that the hardest part of the project was getting the indexing right. I kept facing off-by-one errors especially because whenever I saw a digit, I incremented the loop iterator by that number + 1. When there were two digits, I need to increment it by the corresponding decimal number + 2. One of my friends taught me how to run a debugging session on XCode which made the process a lot simpler.
   2. I also found it strange that the hasProperSyntax function does not check for convertibility (which is done separately.) Initially, I found it difficult to comprehend the difference between the two in the spec so my hasProperSyntax function checked for convertibility as well. I then changed this and moved the code to convertTune.
   3. Finally, I think that a lot of the code that I used for the case of one and two digits is similar which makes me wonder if there a simpler way to account for two digits. However, I was not able to figure a better method quickly enough so I decided to leave it unchanged.
2. **A description of the design of your program:** My program uses two main functions: hasProperSyntax and convertTune but also uses two helper functions to improve readability: 1) isColor and 2) countSlashes

isColor takes in a character (some character in tune, when implemented) and runs a for loop that iterates through an array containing of the “colors”, returning true if there is a match and false otherwise.

countSlashes takes a string (usually a substring of tune uptil some character, when implemented) and runs a for loop that increments a count variable for every occurrence of ‘/’. It then returns the value of count.

**hasProperSyntax** checks that each character in tune is:

either a slash

or a color followed by:

a slash or

a digit followed by:

a slash or

a digit and a slash (unless the first digit itself has a digit before it)

**convertTune** first checks for convertibility and returns errors accordingly. Else, it converts the tune. Checking for errors:

return 1 if tune does not have proper syntax (using hasProperSyntax)

if a sustained note is played, iterate through the next num(digit) char and return 2 if one of them is not a slash (set badBeat to the number of slashes before that + 1)

return 3 if the number is 0, 1, 00 or 01. (set badBeat to 0 or 1 accordingly) *// I was unsure whether to extend this error to 00 and 01 as well but I decided to do it because it seems reasonable if 0 and 1 are meant to induce errors*

return 4 if at least those many num(digit) more chars don't exist (but only return 4 if all the remaining characters are slashes, otherwise we might have to return 2) (set badBeat to countSlashes(tune) + 1)

converting the tune to instructions:

if the function survives all convertibility tests, set instructions = “”

repeatedly (while i is less than string length)

*//\*\*while adding tune[i] to instructions, add lower and upper according to sustained or not*

if tune[i] is a slash (in isolation), add ‘x’ to instructions and increment i

if tune[i] isColor

if next is a slash, add tune[i] to instructions and increment i by 2

if next is a digit define an int amt where it converts the digit(s) to an int

(if branch handles 1 digit and 2 digit separately)

use a for loop to add amt number of tune[i] to instructions

increment i by amt + 2/3 (based on whether its 1 or digit)

1. **Listing all test cases (done using the assert style).** All test cases in the main code have brief comments about the rationale for implementing them, so that block of code has been directly copy pasted here.

assert(hasProperSyntax("G/g/R/r/Y/y/B/b/O/o/")); // checking all colors

assert(hasProperSyntax("")); // no beats

assert(hasProperSyntax("////")); // empty beats

assert(hasProperSyntax("g3///")); // color followed by correct number

assert(hasProperSyntax("g9///")); // color followed by incorrect number

assert(hasProperSyntax("g03/")); // color followed by a two digit number

assert(hasProperSyntax("g0/")); // 0 is syntactically correct, just not convertible

assert(hasProperSyntax("g0/g1/g01/g00//g9//Y3////")); // checking all of the above together

assert( ! hasProperSyntax("g/z//")); // checking for a letter out of color range

assert( ! hasProperSyntax("r")); // checking for a color without a slash

assert( ! hasProperSyntax("r/ g/")); // space should cause an error

assert( ! hasProperSyntax("Password:Nakk\*123%")); //checking random stuff

assert( ! hasProperSyntax("r100/")); // checking 3 digits

string instrs;

int badb;

badb = -999; // so we can detect whether this gets changed

assert(convertTune("", instrs, badb) == 0 && instrs == "" && badb == -999); // empty string returns empty string

assert(convertTune("/", instrs, badb) == 0 && instrs == "x" && badb == -999); // slash returns x

assert(convertTune("////", instrs, badb) == 0 && instrs == "xxxx" && badb == -999); // empty slashes return x's

assert(convertTune("G/g/R/r/Y/y/B/b/O/o/", instrs, badb) == 0 && instrs == "ggrryybboo" && badb == -999); // checking all colors

assert(convertTune("g3///", instrs, badb) == 0 && instrs == "GGG" && badb == -999); // sustained note works correctly

assert(convertTune("G/", instrs, badb) == 0 && instrs == "g" && badb == -999); // upper goes to lower

assert(convertTune("g03///", instrs, badb) == 0 && instrs == "GGG" && badb == -999); // works for 2 digits

assert(convertTune("g11///////////", instrs, badb) == 0 && instrs == "GGGGGGGGGGG" && badb == -999); // works for big 2 digit numbers as well

assert(convertTune("g02//Y2//o///", instrs, badb) == 0 && instrs == "GGYYoxx" && badb == -999); // sustained notes along with empty beats

assert(convertTune("r10///////////y03///O/", instrs, badb) == 0 && instrs == "RRRRRRRRRRxYYYo" && badb == -999); // a lot of random things, checking for any off-by-one errors

assert(convertTune("r//y/g3///o/", instrs, badb) == 0 && instrs == "rxyGGGo" && badb == -999); // checking the last example from the spec

instrs = "WOW"; // so we can detect whether this gets changed

badb = -999; // so we can detect whether this gets changed

assert(convertTune("r", instrs, badb) == 1 && instrs == "WOW" && badb == -999); // something syntactically incorrect returns 1, leaves instrs and badb unchanged

assert(convertTune("o/r3//y/", instrs, badb) == 2 && instrs == "WOW" && badb == 4); // non-slash in a sustained note

assert(convertTune("o/r03//y/", instrs, badb) == 2 && instrs == "WOW" && badb == 4); // non-slash in a sustained note

assert(convertTune("r0/", instrs, badb) == 3 && instrs == "WOW" && badb == 0); // not convertible if the digit is 0,1,00,01, badbeat returns one's digit

assert(convertTune("r1/", instrs, badb) == 3 && instrs == "WOW" && badb == 1); // not convertible if the digit is 0,1,00,01, badbeat returns one's digit

assert(convertTune("r00/", instrs, badb) == 3 && instrs == "WOW" && badb == 0); // not convertible if the digit is 0,1,00,01, badbeat returns one's digit

assert(convertTune("r01/", instrs, badb) == 3 && instrs == "WOW" && badb == 1); // not convertible if the digit is 0,1,00,01, badbeat returns one's digit

assert(convertTune("r3//", instrs, badb) == 4 && instrs == "WOW" && badb == 3); // ends prematurely, add num(slashes) + 1

assert(convertTune("r03//", instrs, badb) == 4 && instrs == "WOW" && badb == 3); // ends prematurely, add num(slashes) + 1

assert(convertTune("r03//Y1//", instrs, badb) == 2 && instrs == "WOW" && badb == 3); // returns the leftmost error