1. I faced many challenges during this project and was constantly stopped by errors both syntactical and logical. One of the first problems I encountered was getting undefined behavior for my string. This occurred because for some of my if statements would try to check a value at a negative index. Also, I had a hard time with getting the brackets around a chord. Since instructions already had values adding a bracket in front of a chord string was very difficult. So I realized I needed to use a temporary variable to store pieces of strings for a while and then reset it to an empty string after adding the contents of temp to instructions.
2. Pseudocode:

…

If it has the Correct Syntax

The result is set to true

Repeatedly:

A counter is made to add one every time there is a beat

If octave is between 0-9

If two accidentals in a row

If spaces in song

If wrong order of the notation e.g. G5#

Result is set to false

Returns true if correct syntax and false if not

…

…

If the song string is a playable string

The answer is set to true

If the string does not have correct syntax, it is automatically false

If the song string does have correct syntax

Repeatedly:

changes C6 to B#5

changes B#1 to C2

changes Cb6 to B5

Repeatedly:

Answer is false if the octave is not between 2 and 5

if the note is Cb2 this note is note playable and answer is false

returns if the string is playable

…

Counts the number of accidentals in a song

Repeatedly

If the character is b or #

Increment accidentals

Return how many accidentals

…

Counts the number of octaves in a song

Repeatedly

If the character is a digit

Increment octaves

Return how many octaves

Encodes the song into instructions if there is no bad Beat and has correct syntax

Integer value is set to 0

Integer beats is set to 1

Integer of the number of notes in a beat is set to 0

Is not correct syntax

Value is 1

Is correct syntax but not playable

Repeatedly:

If character is /

Beat plus 1

If not playable note

Bad beat equals beat

Break

Value is 2

Repeatedly:

If double /

Instruction adds a space

If the character is an uppercase letter

If note only has accidental sign

Encodes a note with octave four, note letter, and sign

Adds one to number

If note has sign and octave

Changes octave to an integer

Encodes a note with the octave given, note letter, and the sign

Adds one to number

If has octave but no sign

Changes octave to an integer

Encodes a note with octave given, note letter, and space

Adds one to number

No sign or digit

Encodes default 4 octave, letter, and space

If character is /

Number goes back to 0

If next character is /

number is more than 1

Instruction adds temp value with []

…

1. Test Cases:

encodeSong("D5//D/", instrs, badb) == 0 && instrs == "R H" && badb == -999);

instrs = "xxx"; badb = -999; // so we can detect whether these get changed

encodeSong("D5//Z/", instrs, badb) == 1 && instrs == "xxx" && badb == -999);

encodeSong("D5//D8/", instrs, badb) == 2 && instrs == "xxx" && badb == 3);

encodeSong("C/C/G/G/A/A/G/", instrs, badb) == 0 && instrs == "GGLLQQL");

encodeSong("D3/F#3/A3/D4//D3F#3A3D4/", instrs, badb) == 0 && instrs == ".8DH [.8DH]");

encodeSong("G3B3DD5//G/A/A3B/C5/B3D5//G//G//CE5//C5/D5/E5/F#5/B3G5//G//G/", instrs, badb) == 0 && instrs == "[SFHR] LQ[DW]E[FR] L L [GT] ERT\*[FU] L L");

encodeSong("DADDA/", instrs, badb) == 0 && instrs == "[HQHHQ]");

encodeSong("//", instrs, badb) == 0 && instrs == " ");

encodeSong("", instrs, badb) == 0 && instrs == "");

Other test cases I used to check for correct syntax includes:

HasCorrectSyntax(“C##”)

HasCorrectSyntax(“8C”)

HasCorrectSyntax(“D5#”)