1. The most notable obstacle I had to overcome was understanding the spec in general, knowing what is syntactically correct, what is convertible, and what is a bad beat. Also, while I was working incrementally, I had to organize my thoughts so that my new code would not disturb the previous code in anyway. Lastly, keeping track and counting the character positions was also a frequent challenge while developing this program.
2. Pseudocode

bool function hasProperSyntax:

if tune is empty 🡪 syntactically correct

set int i to 0

The while loop goes for until i < size of tune {

Nested inner while loop for beats that are not just ‘/’ {

//This allows us to examine the parts of the tune one by one

If tune is not one of the color alphabets 🡪 return false

Else 🡪 Move on to the next character

If the index is greater than or equal to tune size then return false

//This blocks the function from relying on undefined

//behavior

Switch:

Case where digit (0-9) 🡪 Move onto next character then another switch case

If the index is greater than or equal to tune size then return false

//Blocks undefined behavior

Case where / 🡪 breaks and moves onto next part of the tune

Default 🡪 return false; //Anything other than digit or / is false

The next switch case:

Case where digit (0-9) 🡪 Move onto next character then another switch case

If the index is greater than or equal to tune size then return false

//Blocks undefined behavior

Case where / 🡪 breaks and moves onto next part of the tune

Default 🡪 return false; //Anything other than digit or / is false

The final switch case:

* //At this point it needs to be only a slash
* Case where / 🡪 breaks and moves onto next part of the tune
* Default 🡪 Return false; //Anything other than a slash is false

}

}

return true; //When everything passes it means it’s syntactically correct

int convertTune:

if tune is not syntactically correct 🡪 return 1;

//At this point tune is syntactically correct

Set up variables (string translation, int beatnumber, char color)

Set int i to 0

The while loop goes for until i < size of tune {

Set default int sustain to 1

Check if the beat is just a slash

* Then add to translation ‘x’
* Increase beat number by 1
* Move onto next character in string tune
* Go back to top of while loop by using continue;

Check if beat starts off with color

* Save color alphabet to color variable we made
* Increase beat number by 1

Check if the next character is a digit

* + - Since it is a digit we know it is sustained 🡪 color char to upper

Check if the next next character is a digit

* Change digit characters to int then multiply first digit by 10 and add second digit to it. Then save it to int sustain.

Now check if there are correct amount of slashes

Initialize countbeat variable to return correct bad beat if there are incorrect amount of slashes

* For loop (from the first slash; to the supposedly correct index where the last slash should be; increment counter by 1)
  + If one of them doesn’t equal ‘/’
    - If counter for for loop is greater than or equal to tune size then return 4 and bad beat = beatnumber + countbeat as it ended prematurely
    - Else
    - badBeat = beatnumber(current beat) + countBeat(Number of slashes)
    - return 2

Increment countBeat by 1 every for loop

* If sustain is less than 2
  + badBeat = beatnumber (currentBeat);
  + return 3

//At this point the tune is convertible

* Add sustain -1 to beat number

//Sustain -1 as we counted the color letter as a beat

* Add to translation color letters sustain number of times
* Index counter i += 3 + sustain (as it skips the letter then the two digits (3) then the sustain number of slashes)

else //means it is a single digit

* Change digit character to int then save it to int sustain.

Now check if there are correct amount of slashes

Initialize countbeat variable to return correct bad beat if there are incorrect amount of slashes

* For loop (from the first slash; to the supposedly correct index where the last slash should be; increment counter by 1)
  + If one of them doesn’t equal ‘/’
    - If counter for for loop is greater than or equal to tune size then return 4 and bad beat = beatnumber + countbeat as it ended prematurely
    - Else
    - badBeat = beatnumber(current beat) + countBeat(Number of slashes)
    - return 2

Increment countBeat by 1 every for loop

* If sustain is less than 2
  + badBeat = beatnumber (currentBeat);
  + return 3

//At this point tune is convertible

* Add sustain -1 to beat number

//Sustain -1 as we counted the color letter as a beat

* Add to translation color letters sustain number of times
* Index counter i += 2 + sustain (as it skips the letter then the one digit (2) then the sustain number of slashes)

else // Not a sustained note

* Add a lower case color character to translation
* Index counter i += 1 + sustain (as it skips the letter then (1) then the one slash that follows (default sustain is 1)
* Beatnumber += sustain – 1 as we counted color letter being counted as beat

//At this point tune is converted and now we save our translation string to instruction parameter

instruction = translation

return 0 //as it is convertible

}

assert(hasProperSyntax(""));

//Empty tune should be syntactically correct

assert(hasProperSyntax("g/b//"));

//Checks if hasProperSyntax works correctly for syntactically correct tunes

assert( ! hasProperSyntax("g/z//"));

//Checks if hasProperSyntax returns false when color inputted is not one of the correct colors

assert(hasProperSyntax("r/"));

//Checks if hasProperSyntax works correctly for syntactically correct tunes

assert( ! hasProperSyntax("r"));

//Checks if hasProperSyntax returns false when beat doesn’t end with a slash

assert(hasProperSyntax("y03///r10//////////"));

//Checks if hasProperSyntax works correctly for syntactically correct tunes

assert(hasProperSyntax("G/"));

//Checks if hasProperSyntax works correctly for syntactically correct tunes

assert(hasProperSyntax("r//Y/g3///o/"));

//Checks if hasProperSyntax works correctly for syntactically correct tunes

assert(hasProperSyntax("///"));

//Checks if only slash beats return true for hasProperSyntax function as it is indeed //syntactically correct

assert(!hasProperSyntax("G12G/"));

//Checks if hasProperSyntax returns false when inputted beat with three digits

assert(!hasProperSyntax("y033///r10//////////"));

//Checks if hasProperSyntax returns false when inputted beat with three digits when there are multiple beats

assert(!hasProperSyntax("y03///r100//////////"));

//Checks if hasProperSyntax returns false when inputted beat with three digits when there are multiple beat and the syntactically incorrect tune comes later on

assert(!hasProperSyntax(" "));

//Checks if hasProperSyntax returns false when inputted beat with only spaces as it is syntactically incorrect

assert(hasProperSyntax("/////g12//////"));

//Checks if hasProperSyntax works correctly for syntactically correct tunes

assert(!hasProperSyntax("G"));

//Checks if hasProperSyntax returns false when beat doesn’t end with a slash

assert(hasProperSyntax("G1/"));

//Checks if hasProperSyntax works correctly for syntactically correct tunes

string instrs;

int badb;

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

assert(convertTune("r//g/", instrs, badb) == 0 && instrs == "rxg" && badb == -999);

//Checks if convertTune correctly translates tune and doesn’t change badBeat number

assert(convertTune("r/y//g/r/", instrs, badb) == 0 && instrs == "ryxgr" && badb == -999);

//Checks if convertTune correctly translates tune and doesn’t change badBeat number

assert(convertTune("r/y02//g/r/", instrs, badb) == 0 && instrs == "rYYgr" && badb == -999);

//Checks if convertTune correctly translates tune when sustained and doesn’t change badBeat

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);

//Checks if converTune correctly returns 1 when tune is syntactically incorrect

assert(convertTune("r/y3//g/r/", instrs, badb) == 2 && instrs == "WOW" && badb == 4);

//Checks if convertTune correctly returns 2 when the tune is syntactically correct but while a sustained note is in effect, a beat not consisting of only a slash is present; also checks if the correct badbeat number is returned and instructions is not changed

assert(convertTune("r/y0//g/r/", instrs, badb) == 3 && instrs == "WOW" && badb == 2);

//Checks if convertTune correctly returns 3 when the tune is syntactically correct but a beat specifies a sustained note of length less than 2; Also checks if badBeat is set to the number of that beat and instructions is not changed

assert(convertTune("r/y4//g/r/", instrs, badb) == 2 && instrs == "WOW" && badb == 4);

//Checks if convertTune correctly returns 2 when the tune is syntactically correct but while a sustained note is in effect, a beat not consisting of only a slash is present; also checks if the correct badbeat number is returned and instructions is not changed

assert(convertTune("r///y4//g/g/r/", instrs, badb) == 2 && instrs == "WOW" && badb == 6);

//Checks if convertTune correctly returns 2 when the tune is syntactically correct but while a sustained note is in effect, a beat not consisting of only a slash is present; also checks if the correct badbeat number is returned and instructions is not changed

assert(convertTune("r2//g2/", instrs, badb) == 4 && instrs == "WOW" && badb == 4);

//Checks if convertTune correctly returns 4 when the tune is syntactically correct but while a sustained note is in effect, the tune ends prematurely; also checks if the correct badbeat number is returned and instructions is not changed

assert(convertTune("r2//////////y2/", instrs, badb) == 4 && instrs == "WOW" && badb == 12);

//Checks if convertTune correctly returns 4 when the tune is syntactically correct but while a sustained note is in effect, the tune ends prematurely; also checks if the correct badbeat number is returned and instructions is not changed

assert(convertTune("r/y2//g01/r/", instrs, badb) == 3 && instrs == "WOW" && badb == 4);

//Checks if convertTune correctly returns 3 when the tune is syntactically correct but a beat specifies a sustained note of two digits where its length is less than 2; Also checks if badBeat is set to the number of that beat and instructions is not changed