1. The hardest part of Project 3 was to distinguish ticks and notes from an entire long string. Since the ticks always start with upper case A-G and end in  ‘/’, I used a for loop to identify any ‘/’ and added an int variable, tickNum, to represent the number of tick the program is in. The for loop stops when all characters in the string are checked. To identify any notes in a tick, I put a while loop inside the for loop. Since a note always start with upper case A-G, I set the condition of the while loop to “the character at position i is not ‘/’. By using these two loops, I was able to convert the notes within a tick and let the program move on to the next tick whenever a ‘/’ is detected.
2. ***bool HasCorrectForm(string song):***

If the song string is empty: true;

Else:

if the song string does not start with uppercase A-G or /: false;

if the song string does not end in /: false;

Repeatedly:

if the character at is uppercase A-G or / or b or #: false;

else if

a digit is followed by an accidental sign: false;

else if

any / is not follwed by uppercase A-G (except the end of the string): false;

If all criteria met, true;

***int convertSong (string song, string& instructions, int& badTick):***

string newSong;

If the song string is not in correct form: return 1

Repeatedly:

if the character is ‘/’: add a space to new song and move to the next character

Repeatedly (if the character is not ‘/’):

find the start of a note (must be uppercase A-G);

find the accidental sign, if any;

find the octave, and the default is 4;

call the convertNote function and get a converted note;

if the converted note is not playable: return 2

if the next character is not ‘/’: add a ‘[‘ to newSong;

add the converted note to newSong;

if the next character is ‘/’ and more than 1 note: add a ‘]‘ to newSong;

After all notes are converted and added to newSong: instructions = newSong

return 0;

1. assert(hasCorrectForm("D5//D/"));

assert(!hasCorrectForm("D5//Z/"));

string instrs;

int badb;

    instrs = "xxx";

badb = -999;

assert(convertSong("D5//D/", instrs, badb) == 0  &&  instrs == "d y"  &&  badb == -999);     instrs = "xxx";

badb = -999;

 assert(convertSong("D5//Z/", instrs, badb) == 1  &&  instrs == "xxx"  &&  badb == -999);      assert(convertSong("D5//D8/", instrs, badb) == 2  &&  instrs == "xxx"  &&  badb == 3);

**Reason:**

1. Check if hasCorrectForm function recognizes correct song strings;
2. Check if hasCorrectForm function recognizes an incorrect song string that contains an out-of-range uppercase letter Z;
3. Check if convertSong function correctly convert a playable song string into a new song string, and returns 0;
4. Check if convertSong function recognizes an incorrect song string that contains an out-of-range uppercase letter Z, and returns 1;
5. Check if convertSong function recognizes a not playble song string that contains a unplayble note “D8”, and returns badTick and 2.

assert(hasCorrectForm("///"));

        assert(!hasCorrectForm("B5///A"));

        string instrs;

        int badb;

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

assert(convertSong("D3/F#3/A3/D4//D3F#3A3D4/", instrs, badb) == 0  &&  instrs == "9Qey [9Qey]"  &&  badb == -999);

        instrs = "xxx"; badb = -999;

assert(convertSong("D3/F3#/A3/D4//D3F#3A3D4/", instrs, badb) == 1  &&  instrs == "xxx"  &&  badb == -999);

        assert(convertSong("G9//C1/", instrs, badb) == 2  &&  instrs == "xxx"  &&  badb == 1);

**Reason:**

1. Check if hasCorrectForm function recognizes that an empty string is a correct song string;
2. Check if hasCorrectForm function recognizes an incorrect song string that does not end in ‘/’;
3. Check if convertSong function correctly convert a playable song string into a new song string, and returns 0;
4. Check if convertSong function recognizes an incorrect song string, in which an octave is followed by an accidental sign, and returns 1;
5. Check if convertSong function recognizes a not playble song string that contains at least one unplayble note, and returns the first badTick(1) and 2.

assert(hasCorrectForm("/A5//G/"));

        assert(!hasCorrectForm("B5/ /A"));

        string instrs;

        int badb;

        instrs = "xxx"; badb = -999;

assert(convertSong("///", instrs, badb) == 0  &&  instrs == "   "  &&  badb == -999);

        instrs = "xxx"; badb = -999;

assert(convertSong(" B5/ /A", instrs, badb) == 1  &&  instrs == "xxx"  &&  badb == -999);

        assert(convertSong("G6/D1/", instrs, badb) == 2  &&  instrs == "xxx"  &&  badb == 2);

**Reason:**

1. Check if hasCorrectForm function recognizes a correct song string;
2. Check if hasCorrectForm function recognize an incorrect song string that has a space in the string;
3. Check if convertSong function correctly convert a playable song string that contains 3 slashes into 3 spaces and returns 0;
4. Check if convertSong function recognizes an incorrect song string, which has a space, and returns 1;
5. Check if convertSong function recognizes a not playble song string that contains at least one unplayble note and returns the first badTick(2) and 2.