1. Notable obstacles:
2. Converting chars to int – covered in lecture
3. Figuring out how to avoid turning a / into an x while already lowercasing a color (skip the slash by adding 2 to k)
4. Allowing multiple empty beats to happen (used a continue statement I found on a C++ database)

For the isTuneWellFormed function:

Define a constant variable to check for the end of a beat

If tune is empty, return true

If the last char in the tune isn’t a /, return false

Repeatedly:

Loop through each character/beat of tune

If the beat has no note corresponding to it,

move on and check again if there is something different

If there is a digit,

move on

If there is another digit,

move on

If the colors don’t match up,

Return false

Anything else,

Move on

If there isn’t a / at the end of the tune

Return false

Anything else(tune might still be going)

Move on

Return true if all conditions are met

For the translateTune function:

Define the errors to return or return 0 is the tune is ok

If tune was never well formed,

Return 1 (NOT\_WELL\_FORMED)

Define a string to build and later pass to the reference variable

Define an integer to set badBeat to if there is one

Define a variable to notify the function of a sustained note

Repeatedly:

If there is a /, it is either an x beat or part of a sustained note,

If neither of these conditions apply, this check does not run again and the program continues on

If there is a sustained note with a single /,

Record the beat number into badBeat

Return error 3 (SUSTAINED\_NOTE\_NOT\_SUSTAINED)

If there is a digit,

Convert the char to an int and add notify function of a sustained note

If there is a second digit,

Multiply the first digit by 10, then add the second digit to the multiple of 10

If the digit of a sustained note is <2,

Record the beat number into badBeat

Return 2 (WRONG\_SUSTAINED\_LENGTH)

Translate the sustained color (lowercase) to uppercase the same number of times corresponding to the digit read

Else,

The beat isn’t sustained and remains as a lowercase note

Avoid counting the end of a note w/ a beat OR the closing beat as x

If the instructions for the tune have been formed and there are still sustained notes,

Record the beat number of the current beat plus one into badBeat

Return 4 (PREMATURE\_END)

If there is no problem translating,

Assign referenced parameter to finalTune’s value

Return 0 (NO\_PROBLEM)

1. For isTuneWellFormed, I used the interactive main routine supplied in the spec. My test cases were:

(empty string) – to check if zero beats would return true

/

// - to see if the function could handle multiple empty beats

r/, g/, o/, y/, b/ - check for colors

3g/// and 13g/- check for digits

\_/ (space and a slash) – to see if function would return false if there were spaces between beats

For translateTune, I used the assert method mentioned in the spec as well:

string instrs;

int badb;

badb = -999;

instrs = "WOW"

These were my test variables for the test cases. If either instrs or badb were changed(when used), the test case failed.

Test Cases

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

instrs = "WOW";

assert(translateTune("5r/////", instrs, badb) == 0 && instrs == “RRRRR” && badb == -999);

assert(translateTune("10r//////////", instrs, badb) == 0 && instrs == “RRRRRRRRRR” && badb == -999);

**Both check for normal notes, sustained notes, sustained notes with digits >=10 and overall function**

**(knows when a tune ends, can read an empty beat and convert it into an ‘x’ WHILE returning 0)**

assert(translateTune("r", instrs, badb) == 1 && instrs == "WOW" && badb == -999);

**Check what happens when a beat doesn’t end with a / (not well formed and returns 1)**

assert(translateTune("r/3y//g/r/", instrs, badb) == 3 && instrs == "WOW" && badb == 4);

**Checks what would happen if a sustained note had a beat that wasn’t a slash(returns 3**)

assert(translateTune("r/123y//g/r/", instrs, badb) == 1 && instrs == "WOW" && badb == -999);

**Checked what would happen if there are more than two digits in a sustained note(return 1)**

assert(translateTune("r/3g//", instrs, badb) == 4 && instrs == "WOW" && badb == 5);

**Checks that even though a tune is well formed, a premature ending to a sustained note is not valid (Return 4)**

assert(translateTune("r/0g/", instrs, badb) == 2 && instrs == "WOW" && badb == 2);

assert(translateTune("r/-1g/", instrs, badb) == 2 && instrs == "WOW" && badb == 2);

**Both of these checked to that a beat with a digit <2 is invalid (Return 2)**

assert(translateTune("R/r/", instrs, badb) == 0 && instrs == "rr" && badb == -999);

**Checks for, even if color is uppercase, it still is translated into instructions if valid.**

assert(translateTune("3r//y/g/3g///0b/", instrs, badb) == 3 && instrs == "WOW" && badb == 3);

**Checks for leftmost occurring problem**

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

**Checks to see if an empty tune is translatable**

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

**Checks to see if an empty beat is translatable**

assert(translateTune("g/a/", instrs, badb) == 1 && instrs == "WOW" && badb == -999);

**Checks for only the specified colors**