Project 3 Report

Obstacles

1. The first major obstacle I overcame was trying to test for the syntactic correctness of the data in function hasCorrectForm. It was very difficult to find a method to check that each individual note fit the correct form. Ultimately, I found a way to test for the four correct conditions of each note through a for-loop and if-else branch structure.
2. The second difficulty I faced was trying to trace and understand the convertNote function. When I understood how it takes in and prints out outputs, I was able to modify my existing structure that processed each correct song note so that each note fed through the convertNote function and was appended onto a new string that I eventually made equal to “instructions.”
3. I had a lot of trouble with out-of-range for loops. I fixed this by adding a condition to each of my loops that, if checked, confirmed that it couldn’t reach out of range.
4. When I finally had the majority of my program finished, my last major obstacle was placing the brackets in the correct order. I figured out a system that calculates where to place the brackets by the number of notes that have passed, as calculated by individual counters in each note condition.

Description of hasCorrectForm:

* The first function checks that an inputted string has the correct form by checking individually against several unique cases, then by counting the number of syntactically correct notes and comparing this number against the number of characters in the song string with slashes removed.

Pseudocode of hasCorrectForm:

bool hasCorrectForm (string s)

{

if(empty string)

return true;

if(string doesn’t end in ‘/’)

return false;

if(string doesn’t start with the letters A-G OR ‘/’)

return false;

// NOTE : Made separate function to test if a character is a capital A-G //

For loop iterating through string:

{

if a character at position i is a letter note followed by another letter note;

add count to “isNote”

}

For loop iterating through string:

{

if a character at position i is a letter note followed by ‘/’;

add count to “isNote”

}

For loop iterating through string:

{

if a character at position i is a letter note followed by an accidental sign # or b;

add count to “isNoteAccidental”

}

For loop iterating through string:

{

if a character at position i is a letter note followed by a digit 0-9;

add count to “isNoteDigit”

}

For loop iterating through string:

{

if a character at position i is a letter note followed by an accidental sign # or b

followed by a digit 0-9;

add count to “isNoteAccidentalDigit”

}

For loop iterating through string:

{

Remove all slashes and add them to a new string called “noSlash”

}

if (sum of characters of syntactically correct notes is equal to size of noSlash)

return true;

else

return false;

}

Description of convertSong:

* The second function checks for all of the playable note conditions, counting each individually while also converting each playable note into its converted version through convertNote(). Through loops and appending, it creates a string that is the converted song, which I set equal to “instructions” if the string successfully completes its full iteration, signaling that the song is playable.

Pseudocode of convertSong:

int convertSong(string s, string& instructions, int& badTick)

int totNote = 0 // This counts the total notes for later use to place

Brackets

if (the input is in correct form)

{

while -- iterates through the whole string

// NOTE if-else tree checks each playable note condition using separate functions, converting these notes into their instructional version //

if (the string is just ‘/’)

set instructions = ‘ ‘;

return 0;

if (the character at i is a playable single note ) // Each of these are a separate function

convert note into instructions;

count note;

Increment by the size of the note (1);

if (the character at i is a playable note followed by a digit)

convert note into instructions;

count note;

Increment by the size of the note (2);

if (the character is a playable note followed by an accidental followed by a digit)

convert note into instructions;

count note;

increment by the size of the note (3);

if (the character is a playable note followed by an accidental)

convert note into instructions;

count note;

increment by the size of the note;

if (the first character of the string is ‘/’)

convert ‘/’ into ‘ ‘ for instructions;

count 1 tick;

increment by 1;

reset note count;

if (there is only one note before a tick)

Bypass without addint brackets; increment by 1;

count 1 tick;

reset note count;

if (there was multiple notes before the first tick)

Insert brackets at the start of the instructions

insert brackets at the end of the current instructions string;

count 1 tick;

reset note count;

if (there are multiple notes before a tick)

insert brackets before and after these notes;

count 1 tick;

reset note count;

if (there is an unplayable note)

keep instructions parameter the same

set badTick to the tick it failed on;

return 2

}

If it manages to iterate through the entire string then:

Set instructions to the converted song string

return 0;

}

else (if the input is not in correct form)

keep all parameters the same;

return 1;

}

|  |  |
| --- | --- |
| Test Data for hasCorrectForm | Reason |
| *zero ticks* | * Checks that hasCorrectForm allows an empty string |
| A/ | * Checks that hasCorrectForm allows a single note |
| A/B/C/D/E/F/G/ | * Checks that hasCorrectForm allows each of the syntactically correct notes A-G |
| A/B/C/D/E/F/G/H/ | * Checks that hasCorrectForm will return false if a letter out of range is used |
| A/B/C | * Checks that hasCorrectForm will return false if there’s no ending tick |
| A0/A1/A2/A3/A4/A5/A6/A7/A8/A9/ | * Checks that hasCorrectForm returns true for letter note - digit structure |
| A#/Ab/ | * Checks that hasCorrectForm returns true for letter note - accidental structure |
| A#0/A#1/A#2/A#3/A#4/A#5/A#6/A#7/A#8/A#9/Ab0/Ab1/Ab2/Ab3/Ab4/Ab5/Ab6/Ab7/Ab8/Ab9/ | * Checks that hasCorrectForm returns true for letter note - accidental - digit structure |
| /// | * Checks that hasCorrectForm returns true for multiple empty ticks |
| A3#/ | * Checks that hasCorrectForm returns false for using the syntactically incorrect order |
| A3C#E//E//F#3A3D4/A3C#E/ | * Should return true |
| C0C0DC0DD/E#FbB#Cb/B#9/ | * Should return true |

|  |  |
| --- | --- |
| Test Data for convertSong | Reason |
| *empty string* | * Checks that it returns 0 and instructions = the empty string |
| / | * Checks that it returns just 0 and instructions = “ “ |
| A/ | * Converts single note into its instruction character |
| A/B/C/D/E/F/G/ | * Checks that it converts single notes correctly without brackets and with right octave |
| A#/Ab/ | * Checks that accidentals are being converted |
| /A#/Ab/ | * Checks that an empty first tick will be converted to a space |
| /ABC/A/ | * Checks that a chord is converted correctly |
| ABC/A/ | * Checks that a chord is converted correctly if it’s the first thing |
| A/B/C/DEFG/ | * Checks that a chord is converted correctly if it’s the last thing |
| A/B//C/D/ | * Checks that an empty tick is converted correctly to a space |
| A/B/Cb2/D/ | * Checks that Cb2 is not playable ( and badTicks is correct) |
| A/B/Cb7/D/ | * Checks that Cb7 is playable |
| A/B/C7/D/ | * Checks that C7 is playable |
| A/B#1/C/D/ | * Checks that B#1 is playable |
| C/C/G/G/A/A/G/ | * Should convert to ttooppo |
| D3/F#3/A3/D4//D3F#3A3D4/ | * Should convert to 9Qey [9Qey] * Checks for multiple playable notes types and ‘//’ |
| G3B3DD5//G/A/A3B/C5/B3D5//G//G//CE5//C5/D5/E5/F#5/B3G5//G//G/ | * Should convert to [wryd] op[ea]s[rd] o o [tf] sdfG[rh] o o |
| DADDA/ | * Should convert to [ypyyp] |