Preethika Kiruveedula

204912895

Project 3 Report

1. I went through quite a few obstacles in the duration of this project. Some of these obstacles include: understanding of the spec, addressing the various conditions of the project, separating beats and notes from the song string, and being able to apply each note through the encode note function due to the order of the parameters. It took me quite a while to even understand the overall task of the project and how to approach it, and even longer to address every single specification. I had some trouble when it came to the extraction the components from the overall string, but after some simplification I managed to figure it out.
2. Pseudocode:

Bool hasCorrectSyntax: tests the syntax of the song string

Defines valid characters

If song is an empty string

Return False

If the end of the song isn’t a ‘/’ or if it starts with an accidental sign

Return False

If it passes both of the above conditions

Repeatedly check every character of the song

Checks multiple syntax conditions

Returns true or false

If it passes all of the above syntax conditions then

Repeatedly checks each character

If the character is one of the characters defined as valid

Return true

Break

If it isn’t

Return false

Return true

encode Note Function does the conversion from entered song to the one understood by the machine, in between functions because it is called in encode song

bool playableNumberNote Function: tests whether they are playable notes in the song

identifies the octave numbers that are always true

returns true

identifies special octave conditions for numbers 1, 2, 6

returns true if it is satisfies the special conditions

if a note has a number that is not defined in the above conditions

returns false

int encodeSong: encodes the song into the encoded version if it fulfills all of the conditions

if hasCorrectSyntax returned false

return 1;

repeatedly

if character is a slash

beatnumber increases by one

if the character is a number

if the playableNumberNote function is false

sets badBeat to beat number plus one and returns 2

if the playableNumberNote function is true

repeatedly

if a string has two slashes in a row

sets the parameters to the encode note function

instructions equal the result of the encode note

if a string has two letter in a row or a letter and a slash

sets parameters

if two letters in a row

note counter is greater than one so

needs to be brackets around the instructions

instructions equals results of encode note

if there is a letter and then a digit and then letter or slash

octave has to be the digit in the song minus 48

sets parameters for encodeNote

if followed by a letter then

note counter is greater than one so

needs to be brackets around the instructions

instructions equals results of encode note

if there is a letter, digit and accidental sign

sets parameters for encodeNote

if followed by a letter

note counter is greater than one so

needs to be brackets around instructions

instructions equals result of encode note

if song successfully compiles return 0

Then there is the int main function which tests multiple cases using the assert function

1. Test Cases:
2. hasCorrectSyntax("A/H/”)

To make sure that a note letter must be one of the seven letters: A B C D E F G

1. hasCorrectSyntax(“A#/Cb/”)

To make sure that # and b are valid accidental signs

1. hasCorrectSyntax(“B6/C3/”)

To make sure that digits are valid characters in the string

1. hasCorrectSyntax(“C#1/CA/”)

To verify that a note can have a accidental sign and an octave number

1. hasCorrectSyntax(“0/CA/”)

To show that a song string can’t begin with anything other than a note letter and a slash

1. hasCorrectSyntax(“B#b/AFG/”)

To show that two accidental signs can’t be next to each other

1. hasCorrectSyntax(“C56/A/”)

To show that there cannot be two digits next to each other

1. hasCorrectSyntax(“B/#C/”)

To show that a note cannot begin with an accidental sign

1. hasCorrectSyntax(“A/1D/”)

To show that a note can’t begin with a digit

1. hasCorrectSyntax(B3CbD//G//A#4D2C5/B#9/”)

To make sure it can handle more complicated song strings

1. hasCorrectSyntax(“///”)

to show that the song string doesn’t need any note letters etc. just beats

1. hasCorrectSyntax(“BBh/AA//”)

Just to check for characters that aren’t valid

1. encodeSong("C3//A/", instrs, badb)

To make sure that the playable note function is working for an always valid octave number and returns 0

1. encodeSong("A5//C9/", instrs, badb)

To make sure that if there is an unplayable note it returns 2 and makes badbeat equivalents to the beat number of the unplayable note

1. encodeSong(“A/B#1/”, instrs, badb)

To show that there is playable note with the octave number 1 and it will return 0 and badbeat will remain unchanged

1. encodeSong(“A/B1/”, instrs, badb)

To show that the octave number 1 is only valid in one situation and it will return 2 and badbeat will change

1. encodeSong(“B/Cb2/”, instrs, badb)

To show that there is a time when the octave number 2 isn’t valid

1. encodeSong("A/Cb6/", instrs, badb)

To show that the octave number 6 is valid for this condition

1. encodeSong(“A/C6/”, instrs, badb)

To show the second time when 6 is a valid octave number

1. encodeSong(“A/B6/”, instrs, badb)

To show that octave 6 isn’t valid for any other condition

1. encodeSong(“AB/C/”, instrs, badb)

To show that this beat will be encoded in brackets because there are multiple notes

1. encodeSong(“//”, instrs, badb)

To show that two slashes will be encoded as two spaces

1. encodeSong(“#/A/” , instrs, badb)

To show that if it is not a valid song string then instructions/badbeat remains unchanged and returns 1