Music Reading

Due Date: July 1st, 2021; 11:59 PM

Introduction:
The goal of this project is to learn how to manipulate and read strings in different ways while also teaching some basics in music theory. The first half focuses on **transposing** and **chords**. The second half combines those ideas in dealing with **note and chord** while also taking into account **time signature.**

For this project, you will be writing in multiple files. You need to submit the **implementation** of your code. The main file is going to be pre-compiled and the file will be called “main.o” The file that you must submit **should be called “functions.cpp”**.

In order to compile to a working executable, you need to type the following code in your terminal:

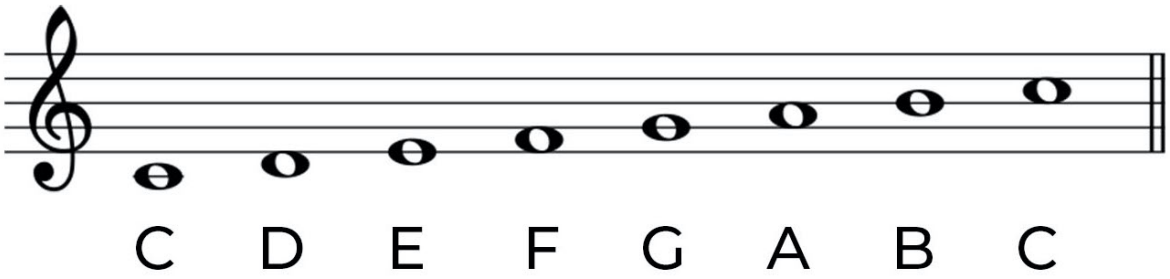
```
g++ functions.cpp main.o
```

This will create an executable file similar to your program. You can run your own main function to test out your results.

You are free to create extra functions to help simplify your workload. Only the functions specified will be tested.

Task A: Transposing Notes

In music composition, all music is composed of notes. A note is a symbol that represents the pitch and duration of a sound. There are 7 notes which can be seen below:



The note that follows after G is actually A, the notes will wrap around indefinitely. However, the first C seen in this image is different from the second C. This is to help indicate the pitch of the note. In this example, the second C is an **octave** above the first C. Feel free to use this website to help you see how the notes are laid out and how they sound: https://www.musicca.com/piano

Visually and auditorily, it is easy to note two identical notes in different octaves. However, in text, it is much harder to notice this. Therefore, we will be using a custom notation for the sake of simplicity:

qA4-qA5-

Let's break this down to one set:
[qA4-]
The first character will indicate the **duration** of the note. The length can be the following:

- [o] indicates **whole note**
- [/] indicates **half note**
- [q] indicates **quarter note**
- [i] indicates **eighth note**

- The next two characters indicate which **pitch** and **octave** are located in. The range of an octave is 1 <= octave <= 8
- The last character is a reserved space to indicate what type of chord the note is. By default, the character will be [-], meaning that it is a single note.

For this task, you must write the following function:

string transposeSong(string notes, int value);

This function takes in a string of notes, and given the value, it will transpose the notes up or down a certain value. The value can be any integer 0 <= value <= 10. If the value of the octave surpasses the range it is supposed to go, it should label that note as "OMIT", but should still transpose the other notes. It should **return the newly transposed notes**.

Example:

String: iC8-/D5-qA2-iC5-/G8-iA6-qE7-oB4-iF1-/C3-qC5-oC1-oC6-
Please enter a value to transpose your notes : 3
Output: iF8-/G5-qD2-iF5-OMITiD6-qA8-oE4-iB2-/F3-qF5-oF1-oF6-

Explanation:

- Let's split our notes into brackets:
- [iC8-] [/D5-] [qA2-] [iC5-] [/G8-] [iA6-] [qE7-] [oB4-] [iF1-] [/C3-] [qC5-] [oC1-] [oC6-]
- For each note, you want to transpose each note UP. If a note goes > G, it should wrap around back to A, and increment the octave by 1.
- In this example, we transpose UP by 3, which gives us the following result:
- [iF8-] [/G5-] [qD2-] [iF5-] [OMIT] [iD6-] [qA8-] [oE4-] [iB2-] [/F3-] [qF5-] [oF1-] [oF6-]

Task B: Reading Chords

A chord is usually three notes that you play simultaneously that harmonize with each other. They are built off a single note, called the **root**. After that, you take the third and fifth note to create a chord.

This is a C chord.

There are a lot more complex rules that determine what **type** of chord it is, which we are **not** dealing with. However, do note that G4-B5-D5 is a valid chord but not G4-B4-D4 is not because (for this project) you begin with the root and go **upwards** for the third and fifth.

This project mentions that the last character is a reserved space. For this task, you must assign the character [c] instead of [-] to indicate that a chord must be played. Not only that, but you must also create the following functions:

void convertAllToChord(string & notes);

- This function will take in all the notes and will convert them to chords.

string convertToChord(string three_notes);

- This function will take in a string that represents three notes. If the notes form a chord, it will return the chord, otherwise return the unmodified string.

This function will take a string of notes, and will convert all the notes to the correct chord and modify the notes string with the chords. You will not be returning **anything**. Instead, it will modify the string passed through.

This is a parameter **passed by reference.** Typically, you would be **passing by value**, which is to say, you create a copy of the contents of the variable, and you are passing it through your function to process. However, passing by reference means that you are referencing the original variable, rather than the copied one. This allows for some interesting properties, one being the fact that by modifying the variable on your function, you are also modifying the variable outside of it. Therefore, you do not have to assign the variable in your main function, you have already done it.

Example:

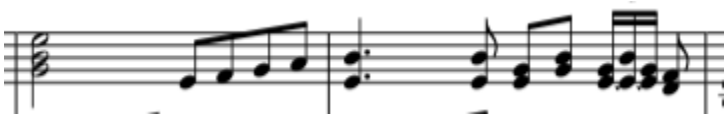
Input:

qA4-qC4-qE4-qG4-qB5-qD5-
iD5-iF5-iA6-iF6-iA7-iC7-/D4-/F4-/A5-
iD1-iF1-iA2-oF6-oA7-oC7-qC4-qE4-qG4-iF6-iA7-iC7-

Output:

qA4cqG4c
iD5ciF6c/D4c
iD1coF6cqC4ciF6c

Task C: Notes and Chords



In practice, a song is not just made of chords. A song is made using a combination of notes and chords. For this task, if a series of notes are a chord, then you must appropriately label them as a chord. Similarly to Task B, you need to take out the notes that belong to the chord, but now, you must preserve the notes that **do not** belong to the chord. Therefore, you must create the following function:

string formatString(const string & to_format);

This will format a string with proper chord labeling and notes in between the chords.

Note: This time, the passed string is not only passed by reference, but it is also const. This is to ensure that the string *to_format* is not modified.

Example:

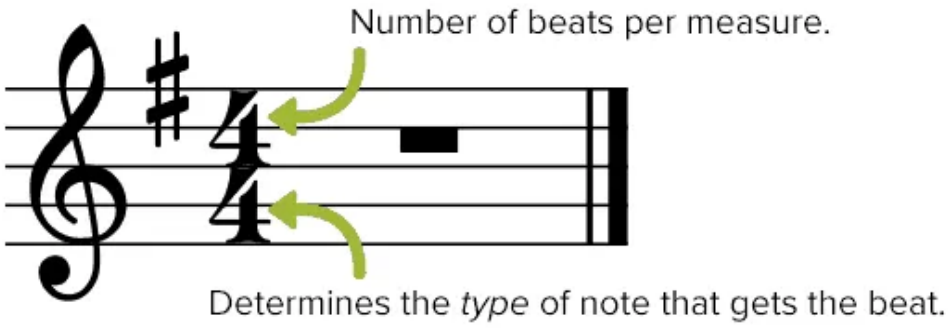
Input:

qA4-qC4-qE4-iB4-iC5-qG4-qB5-qD4-
qB4-qE5-qG5-qB6-oE7-iG3-iB4-iD4-
iE6-iF2-iB5-iE1-iD1-iF1-iA2-qD0-
qF0-qA1-oC1-oE1-oG1-oB4-

Output:

qA4ciB4-iC5-qG4-qB5-qD4-
qB4-qE5ciG3c
iE6-iF2-iB5-iE1-iD1cqD0-
qF0coE1-oG1-oB4-

Task D: Measures



All music is split up into a measure based on the **time signature**. There are multiple time signatures, but we will be using standard time 4/4. What this means, each measure consists of four beats, and a quarter note is the type of note that is the beat. For example, four quarter notes make up a measure, a whole note makes up a measure, 8 eighth notes make up a measure, etc. If the **next note** will increase the measure > 4, it should be considered a new measure.

To increase the complexity of this project, we will be adding yet another symbol to our reserved character and that is [s]. What this means is for that specific note, it will be played on the same **beat** as the previous note. For example:

qA4-qA5s

This means that instead of these two notes taking up two beats, they only take one as the second note is played on the same beat as the first, denoted by [s]. Multiple notes can be played on the same beat. Chords will follow the same rule as the previous tasks (that being that it needs to be three notes with [-] instead of [s])

For this program, you need to create the following functions:

void formatFile(ifstream & song);

What this function does the following:

- Will split the entire song for each measure, all separated by line by line.
- Each measure will be organized as done in Task C.
- The data needs to be exported to a file called “result.txt”.

Do note that the file being taken in is the actual stream, instead of a string line, so feel free to manipulate the stream however you want.

Example:

input:
oE7-oE6siD2-oD0-oF0-oA1-iB2-iB1s/G1-/B2-/D2-iC5-qA7-qC7-qE7-oA7-oF7sqB8-qB5siD3-

Output:
oE7-oE6s
iD2-
oD0c
iB2-iB1s/G1ciC5-qA7c
oA7-oF7s
qB8-qB5siD3-

(Bonus) Task E: Aftermath

After completing this project take a moment to reflect on what you have coded. Are there any oversights on the way data is presented? How would you try to polish up the information to be better displayed?

For this assignment, you must write a one page (double space 12pt font Helvetica Neue) response more or less answering those questions. In addition, you should also explain your approach on the previous tasks, and discuss any setbacks, strengths, and possibly anything you would have done differently in tackling this problem.