Department of Computer Science and engineering

CHIINESE UNIVERSITY OF HONG KONG

Computational identification of chords

Student: Wong Wai Tat (115015727)

Supervisor: Prof. Yip Yuk Lap & Prof. Lucas Wong

Project code: KY1401

Table of Contents

Chapter 1. Introduction

1.1 Background

The last year project had done a piano reduction in music score. This project is an extension from the last year final year project. Piano reduction is a process of both realizing and simplifying full score for multiple instruments. It simplified them into a two- staff piano score. For example, a music score with 4 instruments can be reduced to a music score with two hands piano score. However, decision may need to be made such that the score remain only important notes. Therefore, in the previous year, parameters were set up so that we know which notes are important. In this year, we would like to impose one more feature, which is the harmonic tone recognition. It makes the computer knows which notes form a harmonic tone. Once the harmonic tone is found, we would like to keep those notes. Otherwise, we will see that as redundant notes.

A piano chord is formed by a set of three or more notes. Chords are frequently used in music. When a music instrument plays a chord, it can provide harmonic sounds that contain a feeling of sadness or happiness. In this project, we are going to develop a program to recognize those chords in the music score and implement it to the last year project.

1.2 Objective

The goal of the project is to write a software tool using Python to distinguish which are chords. We have set the following objective for our program:

- Be able to distinguish basic elements. Basic elements like key, note's name, and duration are crucial for a music score since the whole structure of music score is base on those elements.
- Be able to identify the chord. Music score is not only structured by chord, but also structured by many different single notes. We need to find those important chords and filter all redundant notes.
- Be able to report the chord. After a chord is found, we need to report which chord it is with its major.
- Be able to retain only harmonic chord.
- Be able to integrate with the project done last year.

Chapter 2. Basic music element

2.1 Introduction

There are 4 fundamental elements of music, they are:

- Pitch
- Duration
- Timbre
- Loudness

Pitch control highness and lowness of notes such that we can hear sounds different when different pitches apply.

The total time for a note to last depends on its duration. The longer duration time, the longer total time.

Timbre is the quality of sound, which differs by using different music instrument such as violin, trumpet or double bass.

2.2 Clef

Clefs are usually written at the beginning of the score. The composer indicates them such that we will know which musical instrument can be used at the first glance. More importantly, they enable us to tell notes' name through different clef. In the pass history, we got more than 10 kinds of clefs. However, nowadays the common ones are Treble Clef, Bass Clef and Alto Clef. In Figure 2.1, we can see the result of naming different musical notes in Treble Clef and Bass Clef. In Figure 2.2, we can see the result of naming C in Alto Clef.

For Treble Clef, we can see the middle C note is on the newly added bottom ledger line. The notes after C note are D, E, F, G, A … accordingly.

For Bass Clef, we can see the middle C note is on the newly added upper ledger line. The notes after C note are also D, E, F, G, A … accordingly.

For Alto Clef, we can see the middle C note is on the newly added bottom ledger line. The notes after C note are D, E, F, G, A … accordingly.

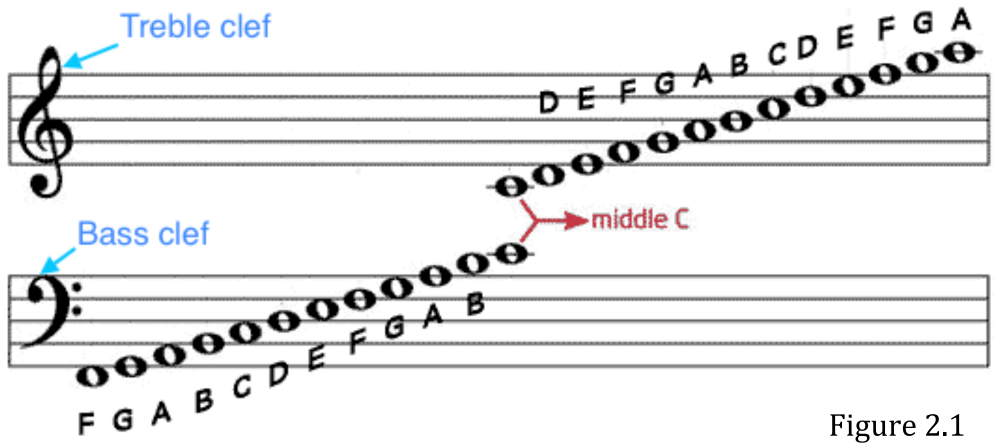


Figure 2.1

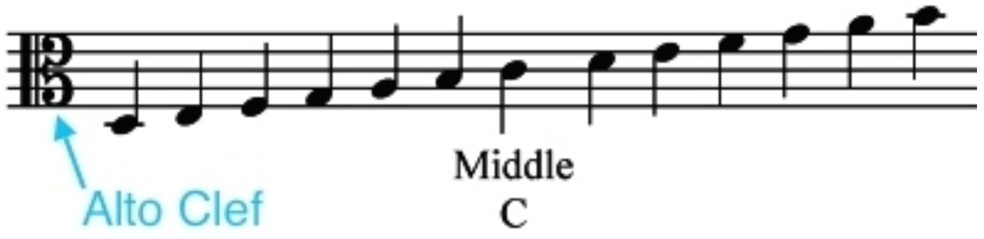


Figure 2.2

2.3 Middle C and Octave

In a music score, we need to find out where the middle C once we know which Clef we are using. Middle C is designated C4 in scientific pitch notation. It is a particular frequency at around 261.6 Hz. By locating where the middle C is, we can know the exact pitch and avoid using the wrong octave. The Figure 2.3 shows where the middle C is on an 88-key keyboard. It is located nearly at the middle of keyboard. Also, we can see where the middle C is in different clef. We use a red line to indicate the middle C such that we can easily compare middle C in different clef.

Furthermore, there are 8 octaves in piano. Each octave contains 7 notes (ie. A-G). We can distinguish which octave the music notes locate by focusing on the subscript of the music note. In Figure 2.3, we can see the C notes located from first to eighth octave.

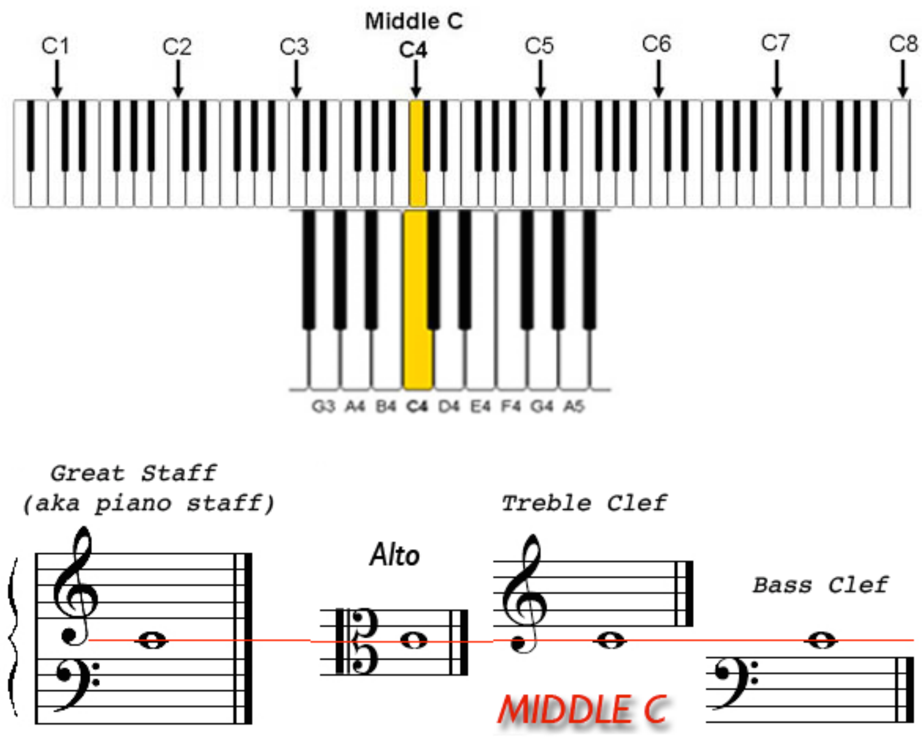


Figure 2.3

2.4 Rhythm value

Every note has their particular duration in the score. A rhythm value is a symbol indicating relative duration. Since the speed of the song depends on another parameter, rhythm symbol only indicate relative duration. Therefore, for the duration a quarter note may differs in different music score. However, we all have a consensus that different types of notes have a general relation. For example, if 1 whole note last for 1 second, than the half note will last for 0.5 second. If 1 whole note last for 2 second, than the half note will last for 1 second. The Figure 2.4 shows the whole architecture of the rhythm value.

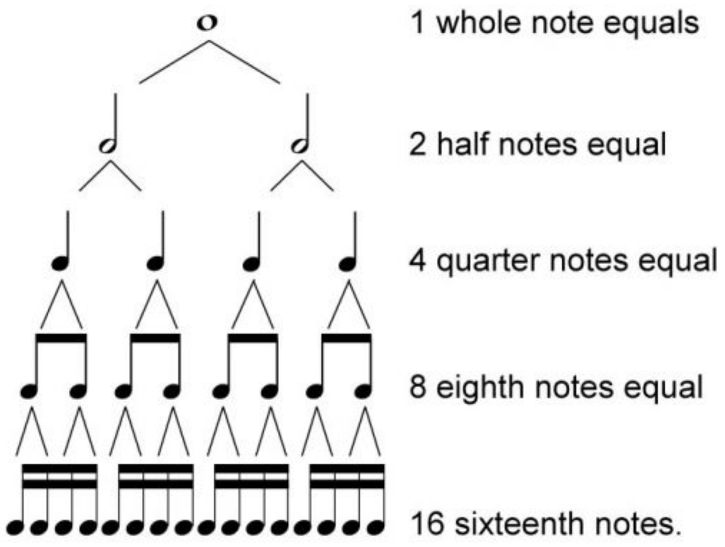


Figure 2.4

2.5 Flat, natural and sharp

Other than notes A, B, C, D, E, F and G, there are several modifiers to adjust the pitch of notes. They are mainly flat, natural and sharp which will be located near the notes and near the clef. The following will explain usage of them:

1. b = flat. It is one half step lower than (left of) a white key.
2. ♮ = natural. It cancels other accidentals; indicates white notes on a piano.
2. # = sharp. It is one half step higher than (right of) a white key.

Chapter 3. Key: Major and minor

3.1 Introduction

The whole texture of a music score is greatly affected by its key. More importantly, major and minor will affect the feeling of a piece of music. If the song is in major, the feeling will become positive, bright and energetic. However, if the song is in minor, the feeling will become negative, dark and sad. In support of Hevner's finding is Meyer's (1956) theory of deviations from expectations. He stated that major key have more regular and normative melodies which can imitate human mood of joy. On the other hand, minor key have a complex and irregular melodies, which can imitate human mood of sadness.
While a survey conducted by Carpuso(1952) to college music instructors by rating musical selections according to corresponding adjectives. The result shows that the majority rate happiness feeling towards a major melody and a majority people rate sadness towards a minor melody. The above survey and theory shows that major/minor melody in music play a crucial role in the history. Even nowadays, we will have the same feelings towards major/minor melodies.

3.2 Major/minor differs in scale

Major and minor differs in the third note. We usually define a melody by finding the difference in 2 pairs of notes. The first pair is 2nd and 3rd notes while the second pair is 5th and 6th notes.

For major scale, we can see Figure 3.1. The pattern of the scale for the first three notes is Whole-Whole-Half. The difference between 2nd and 3rd notes is a whole step. The pattern of the scale for the last four notes is Whole-Whole-Whole-Half. The difference between 5th and 6th notes is a whole step also. Therefore, we can define this scale as a major scale. Then we can sing the scale with Do-Re-Mi-Fa-So-La-Ti-Do, which produce a bright feeling.

For minor scale, we can see Figure 3.2. The pattern of the scale for the first three notes is Whole-Half-Whole. The difference between 2nd and 3rd notes is a half step. The pattern of the sale for the last four notes is Whole-Half-Whole-Whole. The difference between 5th and 6th notes is a half step also. Therefore, we can define this scale as a minor scale. Then we can sing the scale with La-Ti-Do-Re-Mi-Fa-So, which produce a sad feeling.

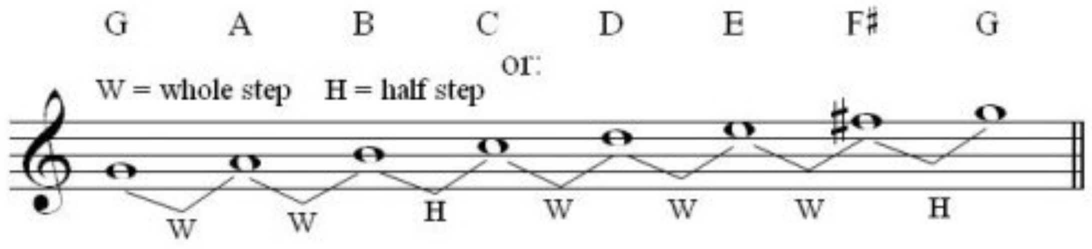


Figure 3.1

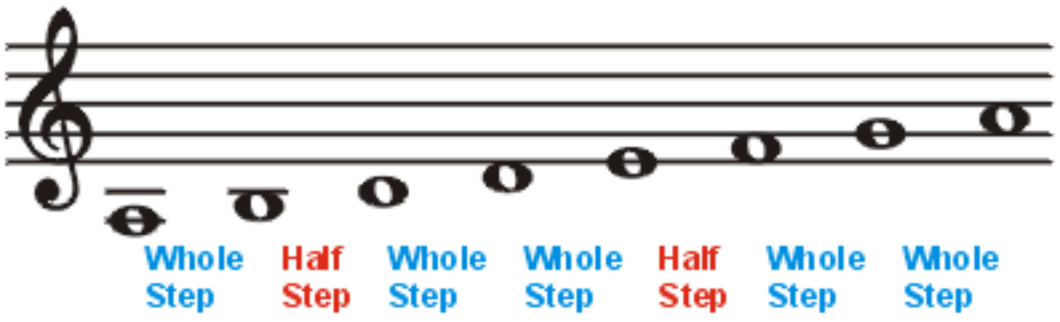


Figure 3.2

3.3 Key signature

The key signature was used in late 17th-century. Before 17th-century, people mark sharp or flat near the notes. However, because of the increasing complex and variety, multiple of sharp or flat symbols are used in a music sheet. It causes a problem that the piece of music sheet becomes messy if they continue to mark sharp or flat near the notes. Therefore, people start to think how to solve this problem. As a result, they mark the sharp or flat symbols near the staff, which is now known as key signature.

Key signature contains a set of sharp or flat symbols placed near the staff. Those symbols indicate that whenever the score is being play, some notes will be played in a slightly different manor. It is either higher for the sharp symbols, or lower for the flat symbols. It affects those notes for the whole piece of music until it reach a double bar line. Figure 3.3 shows the key signature and its corresponding sharp or flat symbols' location.

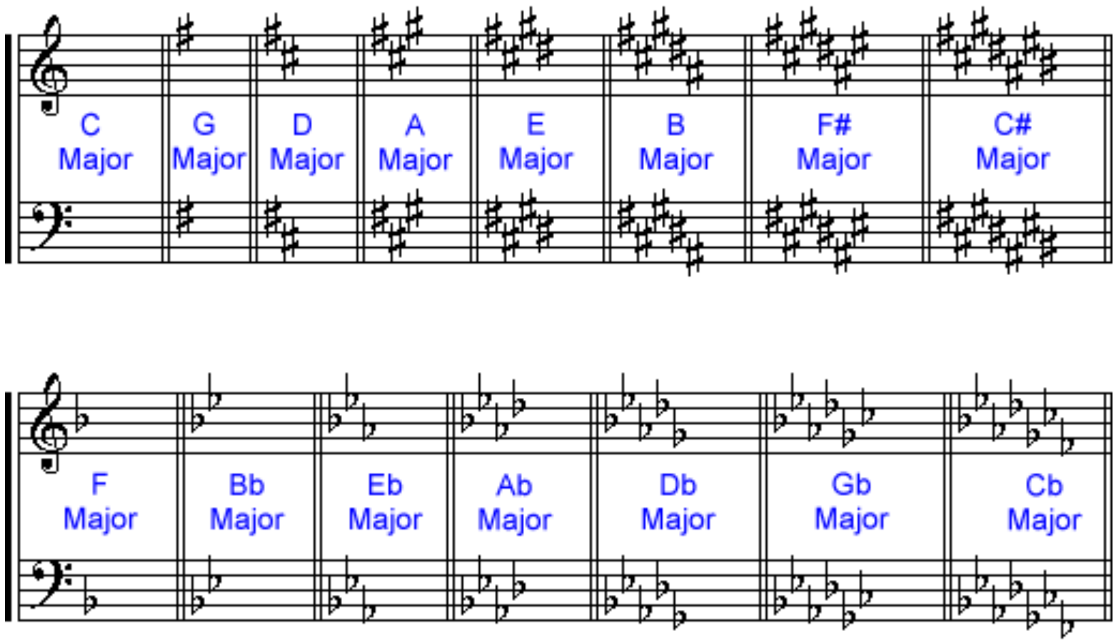


Figure 3.3

3.4 Circle of fifths

The circle of fifths presents relationships between major/minor and key signatures. The 12 tones of chromatic scale relationship can be found in circle of fifths and we can see the major outside the circle and minor key inside the circle. This circle can help composers to build chords and melodies.

It structured in a way that when each key move to the next key, it is always starting on the fifth note of the preceding key. When it moves clockwise, we add one sharp and when it moves anti-clockwise, we add one flat. For example, when C major moves to G major, it adds one sharp. At the same time, we name it as G major due to this pattern: C->D->E->F->G. Another example, when C major moves to F major, it adds one flat. At the same time, we name it as F major due to this pattern: C->B->A->G->F. The Figure 3.4 shows the result.

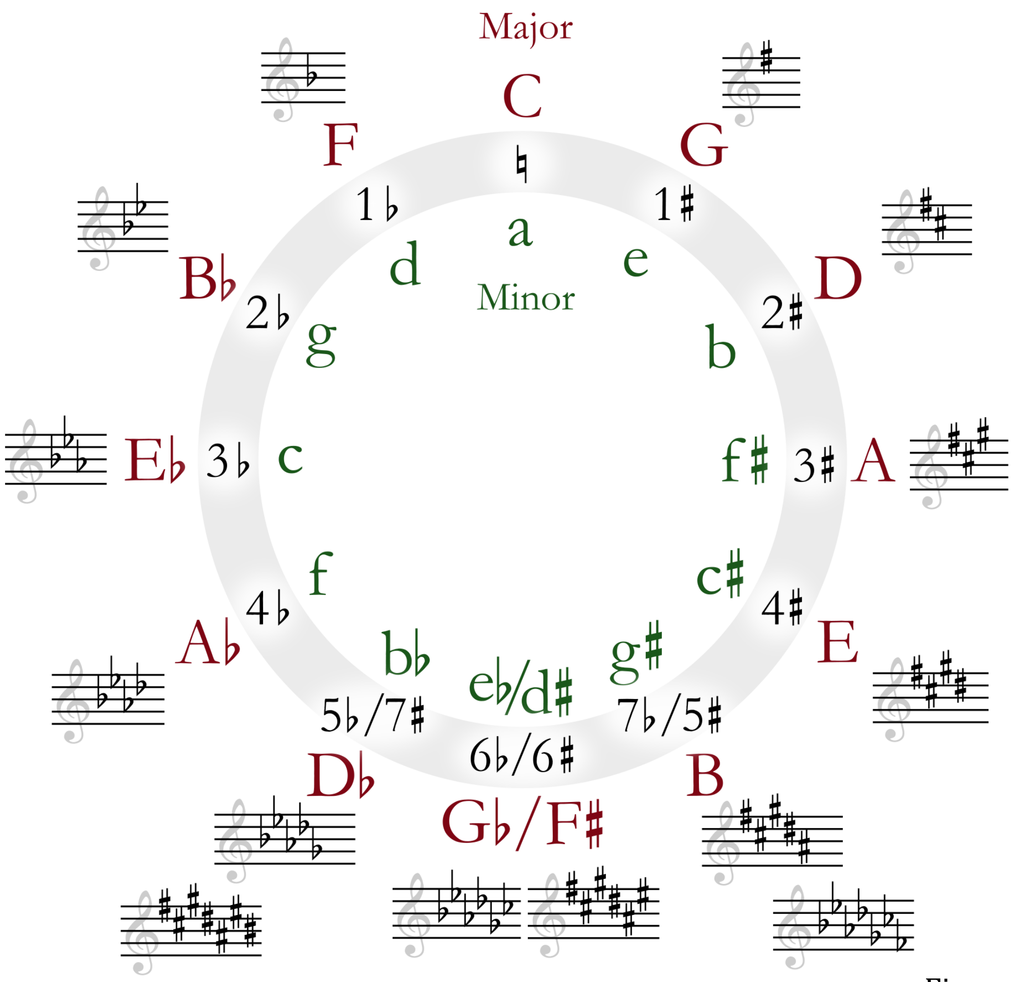


Figure 3.4

Chapter 4. Chords

4.1 Triad chords

Triad chords are used for accompaniment in piano or guitar. If we want o accompany a singer or a violin player, we can use triad chord to help. Those chord make the whole song seems to be harmonic. When the audiences listen to the song, they can get a more comfortable feeling.

A triad means that three-note stacked together and played together at the same time. We called the bottom one as root, the middle one as 3rd and the top one as 5th. The 3rd is a note with 4 semitones higher than the root while the 5th is a note with 3 semitones higher than the 3rd. In Figure 4.1, we can see 3 type of chord including C-E-G, F-A-C, G-B-D. They are major triad chord.

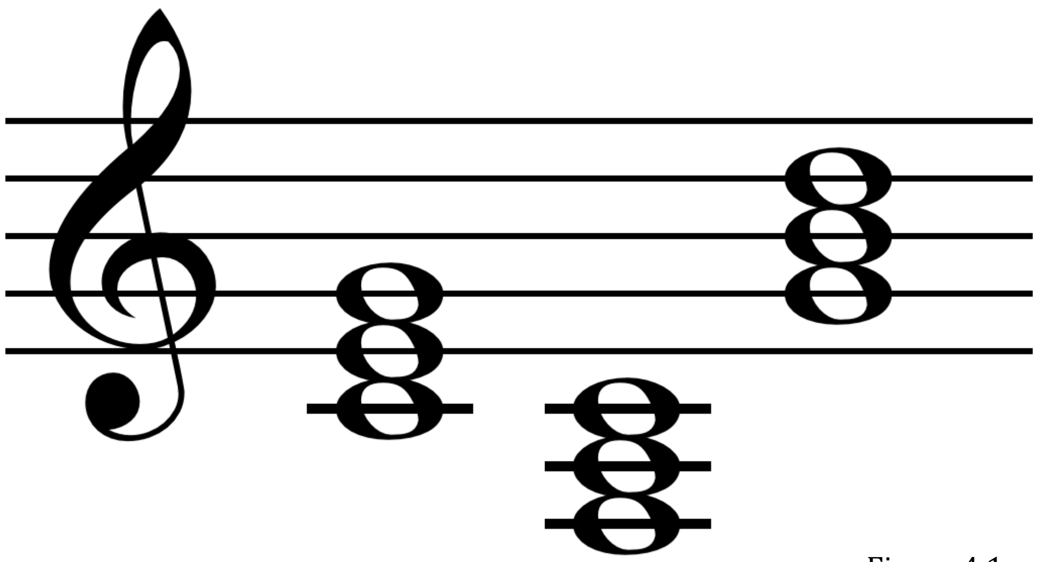


Figure 4.1

4.2 Inversions

The reason we need to use inversion in music is mainly because we can have a more interesting progression. If we keep dwelling a chord for a period of time, it will be boring. Therefore we got inversions to make the music more fun.

Chord inversions descript the position of the three notes, which form a triad chord. They are usually different from the normal situation, which means one of the triad notes may move an octave higher than it is in root position. In Figure 4.2 we find that for a triad chord, it contains C-E-G. The 3 notes can be rearranged such that an inversion occurs. For example, in root position, C is at the bottom. However, if we move the C note an octave higher than it is, then it is called First Inversion with note E at the bottom and G at the middle. If we continue to move the E an octave higher than it is in root position, then it is called Second Inversion with note G at the bottom and C at the middle.

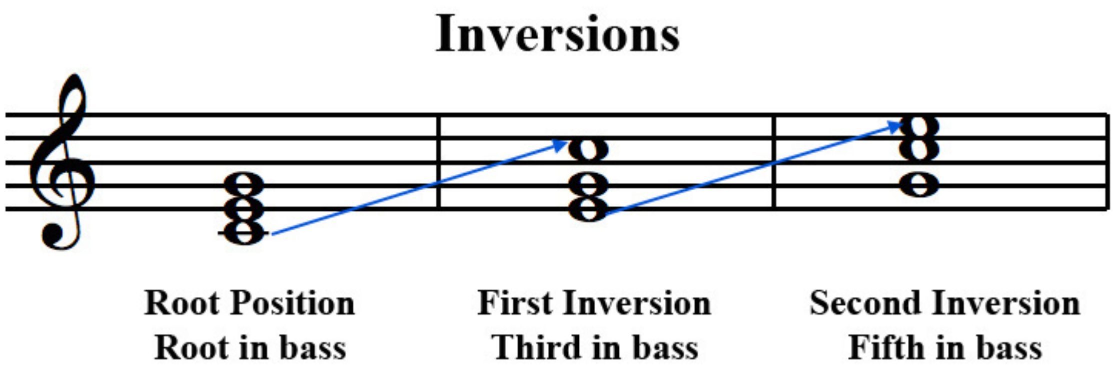


Figure 4.2

4.3 Roman numeral notation

The Roman numeral notation is used in building chords in major/minor scales. Roman number from 1 to 7 names these chords. Although the chord has its own name, we use roman number to name those chord allows us to easily talk about chords progression without depending on the key.

Figure 4.3 shows the way we name a chord with roman number in C major. We can see chord C-E-G is I, D-F-A is II, E-G-B is III, F-A-C is IV etc.

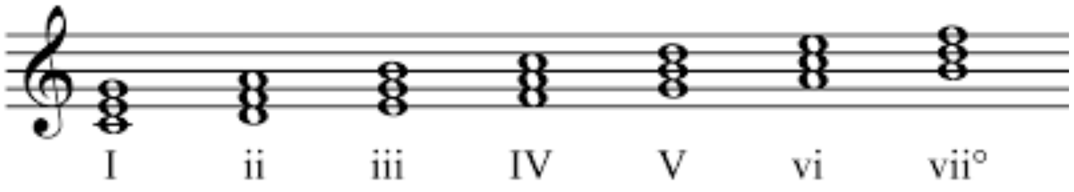


Figure 4.3

Chapter 5. Music XML

5.1 Introduction

With the fast growth of HTML and the Web, XML builds on experience with its predecessor SGML, as well as the experience XML have advantage between simplicity and power. Structured data and musical scores fit that description can be represented by XML. Using XML has no need to worry about the basic syntax of the language, instead letting us worry about what to represent, versus the syntax of how to represent it. There is not necessary to write a low-level parser to read the language since XML parsers are ubiquitous. Music software can reduce the investment in tools made by much larger markets like electronic commerce by basing music interchange language on XML.

In this project, we will mainly focus on the files that are in XML format. The Music XML files are freely available for its spread into the music industry. Many companies, researcher and developers are using it in different product such as MakeMusic, Sibelius or even in iOS apps. This shows how ubiquitous the Music XML is.

Another great things in using this Music XML is that there are plenty of tools for running it. They wrote the Dolet for Finale plug-in in C++ and Java so that the software could run on Mac OS X and Windows. Other software tools are using Java only for development such that there is limitation in some platform.

Also, there are many parsers available in the net. For example, Flash, Flex and Python have the built-in XML support for users who want to do read/write for XML files.

5.2 Overview

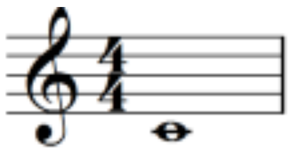
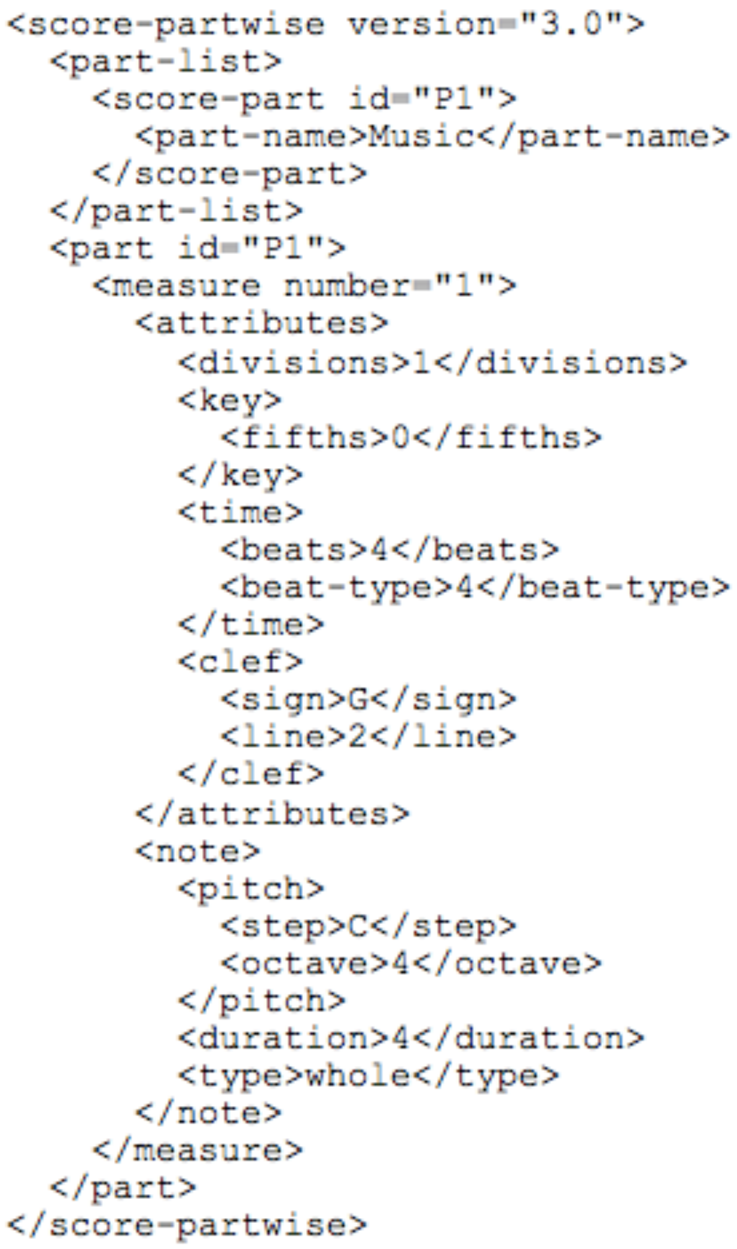


Figure 5.1



```xml
<score-partwise version="3.0">
  <part-list>
    <score-part id="P1">
      <part-name>Music</part-name>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>0</fifths>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
      </note>
    </measure>
  </part>
</score-partwise>
```

Figure 5.2

In Figure 5.1, we got a C note in the score. If we look at the XML file, we got lines of code like the Figure 5.2 shown. We got the key with 0 fifth means that it is a C major with no sharp/flat. We also got the clef with sign G and line 2 indicating it is a Treble clef. Then, we see the pitch of notes. It is a C note with an octave of 4. After that, we can see the duration and type of the note. It is a whole note with duration 4. The XML clearly descript the note on the music score sheet in this case. They are all inside tags such that when we use a parser to read and write the XML file, the parsers can easily locate attribute and read the information it provide.

Chapter 6. Music21

6.1 Introduction

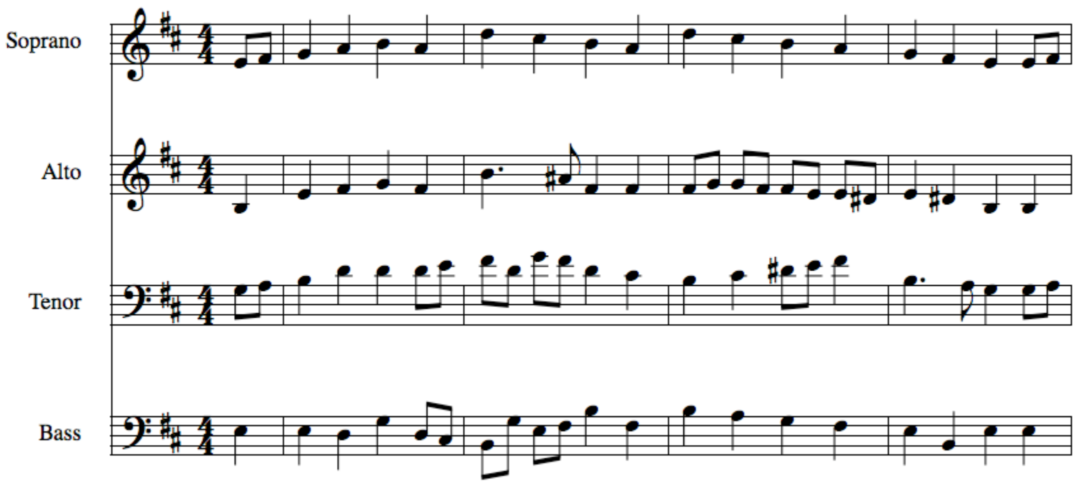
Music21 is a Python-based toolkit for computer-aided musicology. Computational musicology, music information, musical example extraction and generation, music notation editing and scripting are the application of this toolkit. Using music21, Python is able to collect specialized objects and tools. This is a Python extension library. Using a short collection Python code, we can figure out and display a solution simultaneously to many musical problems such as finding the key of the song or figure out the note's name.

6.2 Overview

Example below shows a method to translate a file in XML into a music21 object. One of the method is parse() method. It can copy the XML file to an object. Then the method .show() can act as a command to open the XML file.

```
from music21 import *
sBach = corpus.parse('bach/bwv7.7')
sBach.show()
```

The picture below shows what happen after the command is being entered.



And of course, Music21 can find chords by using some module. However, we find the result came out is not as expected. There were some errors when we used those modules to define whether what chord of certain notes belongs to.

For example, if we get a chord "A#C#E#" in C minor. There should be no such chord in C minor and should be fetching for any other substitute. However, the program shows some strange result. It gives a roman number #vi with inversion #63. Therefore we will implement another logic flow to find the roman number and inversion of the chord.

```
romanNumeral2 = music21.roman.romanNumeralFromChord(
...    music21.chord.Chord(['A#3','C#3','E#3']),
...    music21.key.Key('c'),)
>>> romanNumeral2
<music21.roman.RomanNumeral #vi#63 in c minor>
```

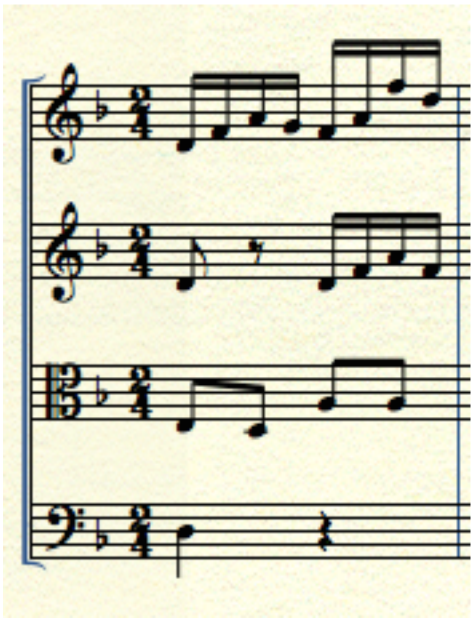
Chapter 7. Identification of chords

7.1a Logic for changing a long duration note to several short duration notes

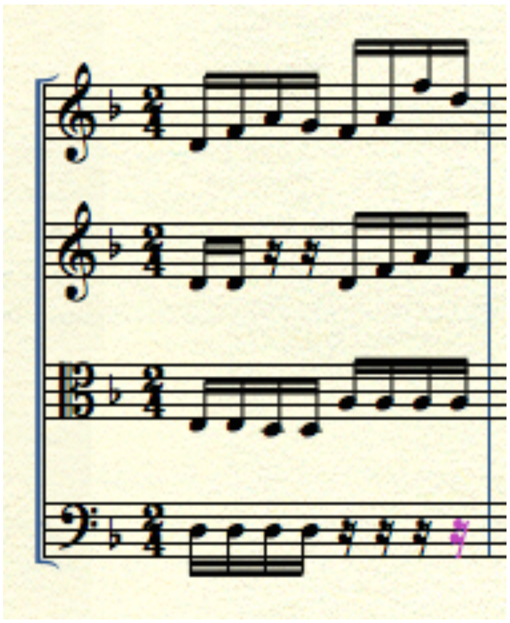
In figure below, we find that the second part of the score contain an eighth note with pitch "D" in the beginning. After that, an eighth rest follows it.

However, in the first part of the score, they are all sixteenth notes. We find that it is difficult for us to do chord comparison. Therefore, we will do a conversion.

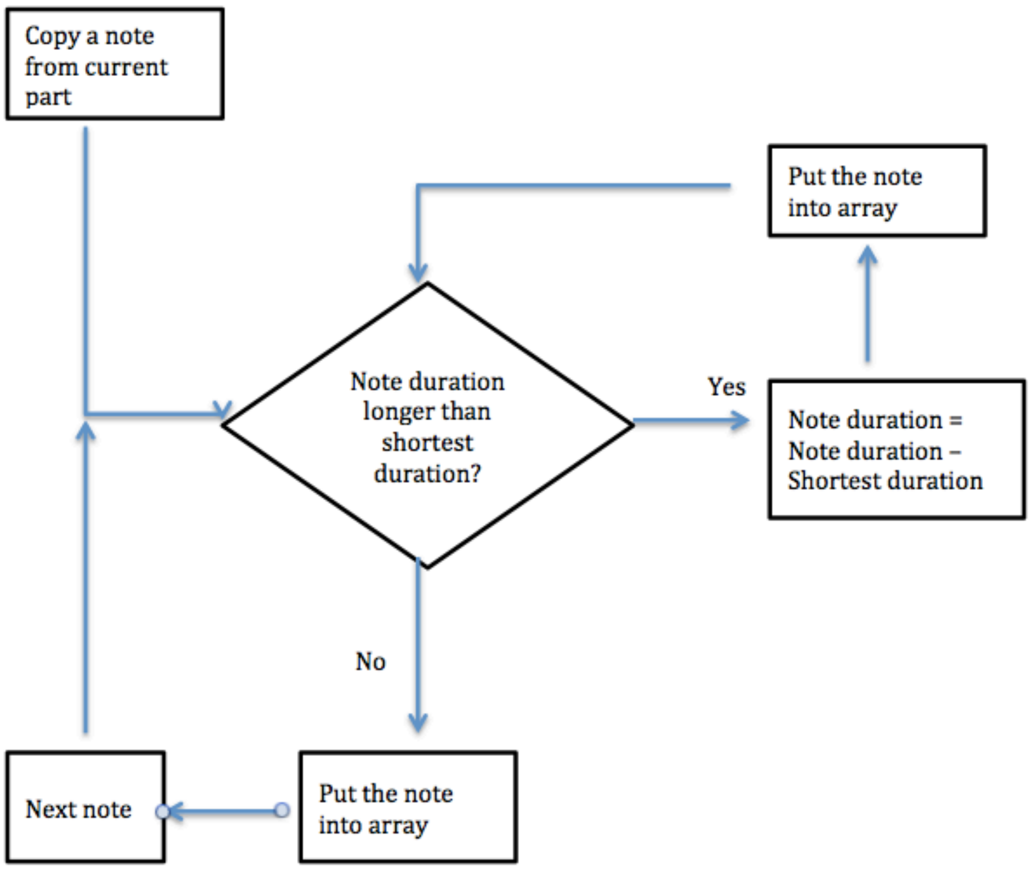
In this case, we want to convert all the notes and rest of the 2nd part of score into shortest duration notes. In other words, we want to convert the notes of the 2nd part of score into sixteenth notes, which have the same duration with the notes of 1st part of score.



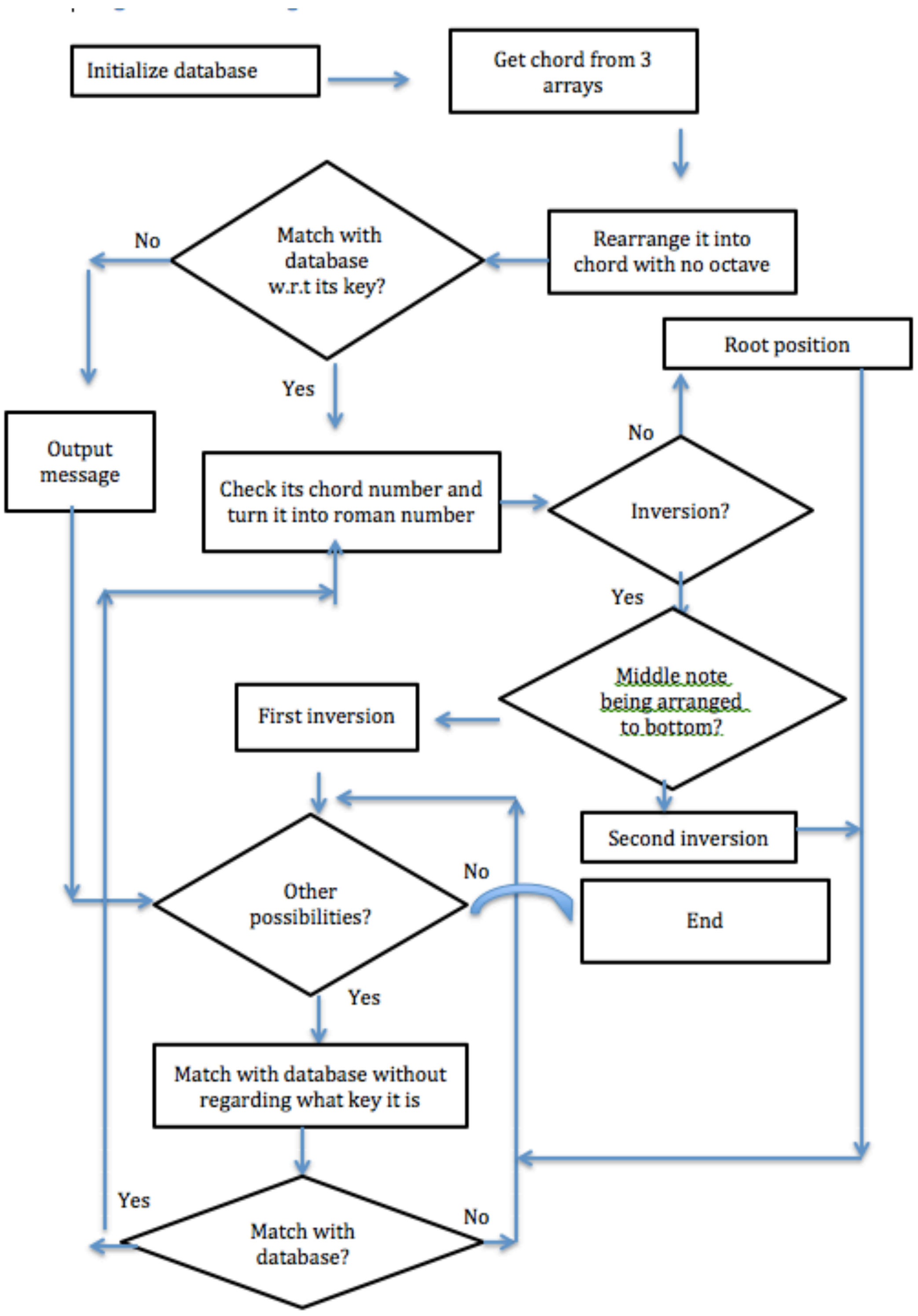
In the figure below, we convert the notes with the shortest duration. This is actually the same with the previous figure. They produce the same harmonic sound. With the figure below, we can easily do the chord comparison. For example, we can easily found out that the 1st four pitches to do comparison is "D"-"D"-"E"-"D". The 2nd four pitches to do comparison is "F"-"D"-"E"-"D".



7.1b Logic flow for changing a long duration note to several short duration notes

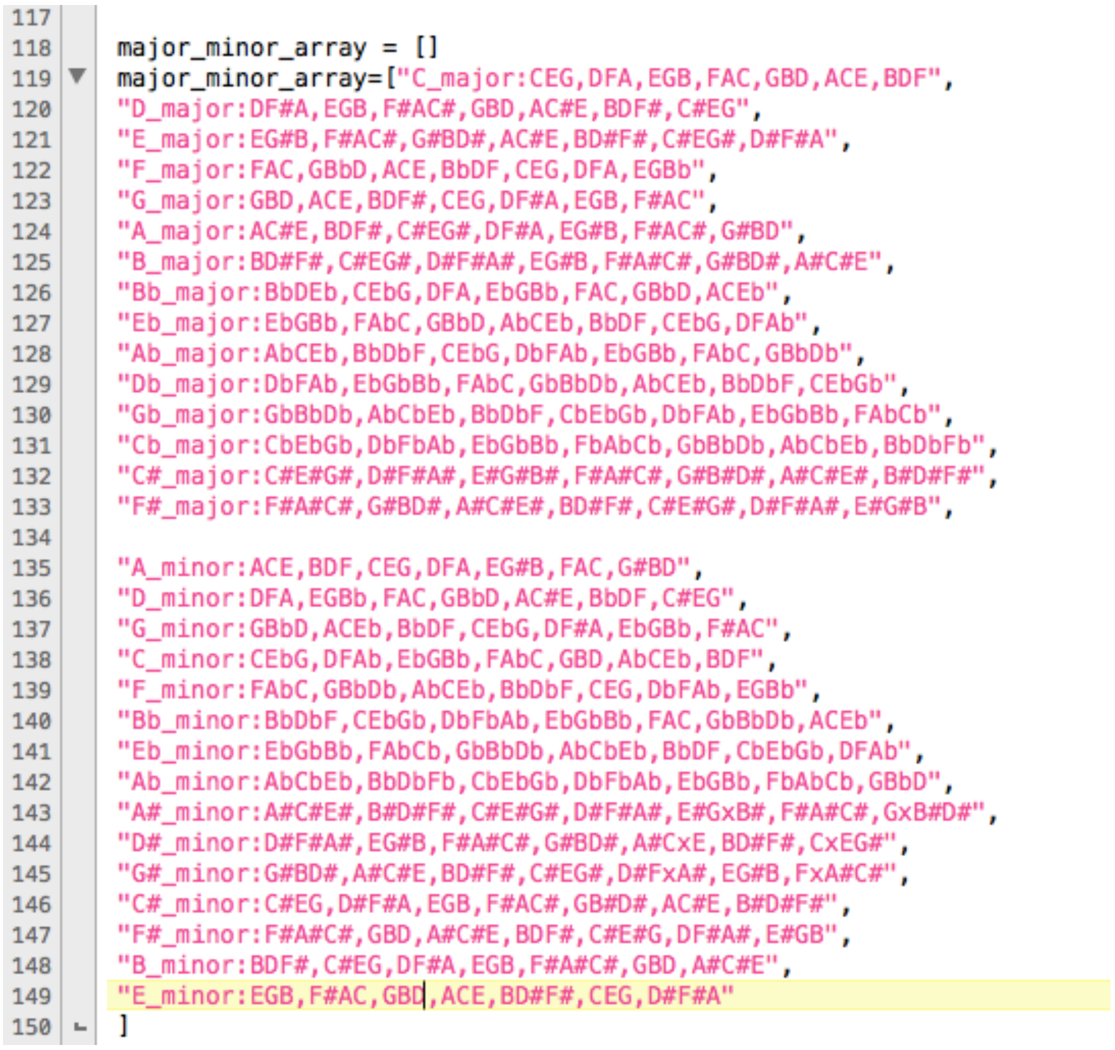

Copy a note from current part
Put the note into array
Note duration longer than shortest duration?
Yes
Note duration = Note duration – Shortest duration
No
Next note
Put the note into array

7.2 Logic flow for finding chords


Initialize database
Get chord from 3 arrays
Rearrange it into chord with no octave
Match with database w.r.t its key?
No
Yes
Output message
Check its chord number and turn it into roman number
Root position
No
Inversion?
Yes
Middle note being arranged to bottom?
First inversion
Second inversion
Other possibilities?
No
Yes
End
Match with database without regarding what key it is
Match with database?
Yes
No

7.3 Database for 7 chords

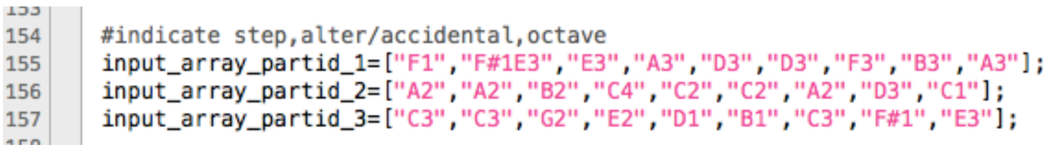
Before inputting any triad chord, we first use an array to store all the 7 chords of the 12 major and minor. By comparing the input chord with the ones in array, then it can find the corresponding chord in roman number.



```
major_minor_array = []
major_minor_array=["C_major:CEG,DFA,EGB,FAC,GBD,ACE,BDF",
"D_major:DF#A,EGB,F#AC#,GBD,AC#E,BDF#,C#EG",
"E_major:EG#B,F#AC#,G#BD#,AC#E,BD#F#,C#EG#,D#F#A",
"F_major:FAC,GBbD,ACE,BbDF,CEG,DFA,EGBb",
"G_major:GBD,ACE,BDF#,CEG,DF#A,EGB,F#AC",
"A_major:AC#E,BDF#,C#EG#,DF#A,EG#B,F#AC#,G#BD",
"B_major:BD#F#,C#EG#,D#F#A#,EG#B,F#A#C#,G#BD#,A#C#E",
"Bb_major:BbDEb,CEbG,DFA,EbGBb,FAC,GBbD,ACEb",
"Eb_major:EbGBb,FAbC,GBbD,AbCEb,BbDF,CEbG,DFAb",
"Ab_major:AbCEb,BbDbF,CEbG,DbFAb,EbGBb,FAbC,GBbDb",
"Db_major:DbFAb,EbGbBb,FAbC,GbBbDb,AbCEb,BbDbF,CEbGb",
"Gb_major:GbBbDb,AbCbEb,BbDbF,CbEbGb,DbFAb,EbGbBb,FAbCb",
"Cb_major:CbEbGb,DbFbAb,EbGbBb,FbAbCb,GbBbDb,AbCbEb,BbDbFb",
"C#_major:C#E#G#,D#F#A#,E#G#B#,F#A#C#,G#B#D#,A#C#E#,B#D#F#",
"F#_major:F#A#C#,G#BD#,A#C#E#,BD#F#,C#E#G#,D#F#A#,E#G#B",

"A_minor:ACE,BDF,CEG,DFA,EG#B,FAC,G#BD",
"D_minor:DFA,EGBb,FAC,GBbD,AC#E,BbDF,C#EG",
"G_minor:GBbD,ACEb,BbDF,CEbG,DF#A,EbGBb,F#AC",
"C_minor:CEbG,DFAb,EbGBb,FAbC,GBD,AbCEb,BDF",
"F_minor:FAbC,GBbDb,AbCEb,BbDbF,CEG,DbFAb,EGBb",
"Bb_minor:BbDbF,CEbGb,DbFbAb,EbGbBb,FAC,GbBbDb,ACEb",
"Eb_minor:EbGbBb,FAbCb,GbBbDb,AbCbEb,BbDF,CbEbGb,DFAb",
"Ab_minor:AbCbEb,BbDbFb,CbEbGb,DbFbAb,EbGBb,FbAbCb,GBbD",
"A#_minor:A#C#E#,B#D#F#,C#E#G#,D#F#A#,E#GxB#,F#A#C#,GxB#D#",
"D#_minor:D#F#A#,EG#B,F#A#C#,G#BD#,A#CxE,BD#F#,CxEG#",
"G#_minor:G#BD#,A#C#E,BD#F#,C#EG#,D#FxA#,EG#B,FxA#C#",
"C#_minor:C#EG,D#F#A,EGB,F#AC#,GB#D#,AC#E,B#D#F#",
"F#_minor:F#AC#,GBD,A#C#E,BDF#,C#E#G,DF#A#,E#GB",
"B_minor:BDF#,C#EG,DF#A,EGB,F#A#C#,GBD,A#C#E",
"E_minor:EGB,F#AC,GBD,ACE,BD#F#,CEG,D#F#A"
]
```

7.4 Input chord format (last semester)

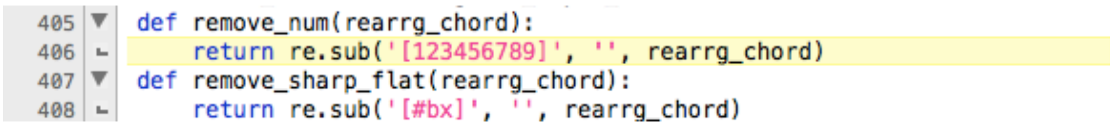
Inputting notes into 3 arrays will simulate the input. Each array contains a number of musical notes. Each note contains a note name, alter/accidental and octave. This is actually the format get from the XML file. We use this to simulate three parts of music score. For example, array1 and array2 simulate the Treble part and Bass part of piano respectively. Array3 will simulate the Treble part of a violin. Then the chord will be save vertically in an array. For example, the first chord of there three array will be FAC, the second will be F#EAC etc.



```
#indicate step,alter/accidental,octave
input_array_partid_1=["F1","F#1E3","E3","A3","D3","D3","F3","B3","A3"];
input_array_partid_2=["A2","A2","B2","C4","C2","C2","A2","D3","C1"];
input_array_partid_3=["C3","C3","G2","E2","D1","B1","C3","F#1","E3"];
```

7.5 Remove sharp or flat symbols and octave

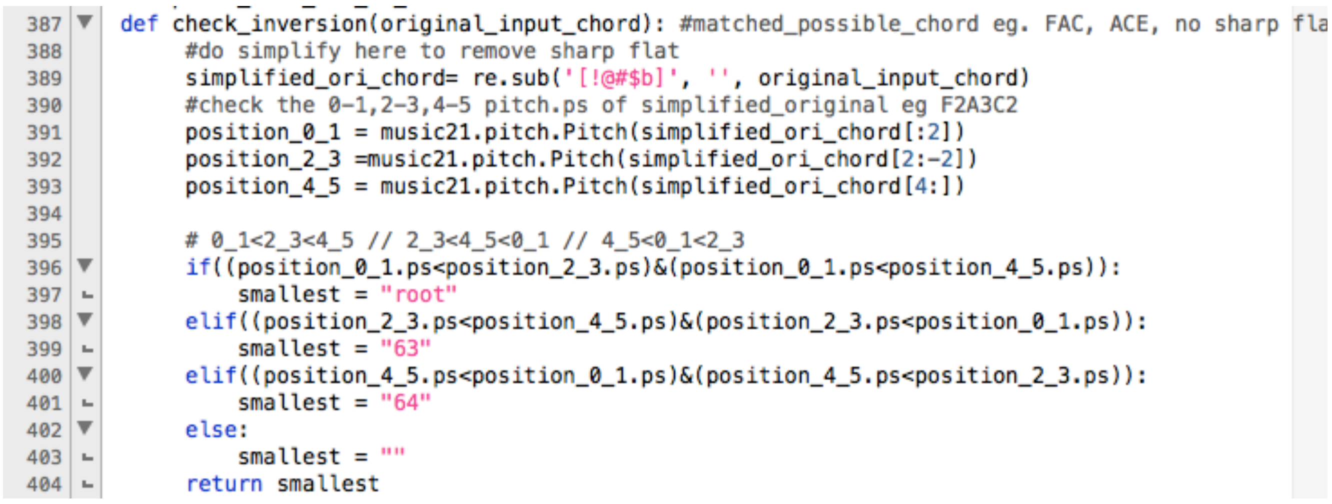
We will then remove sharp of flat symbols for the chord. Then the chord will no sharp/flat will be rearranged by another function so that it can output the corresponding triad chord and match it will the database. For example, if the chord is E3B2G2, then after the removal of sharp or flat symbols and octave, the remaining EBG will be arrange by another function such that it can return a EGB chord and match with the database.



```
def remove_num(rearrg_chord):
    return re.sub('[123456789]', '', rearrg_chord)
def remove_sharp_flat(rearrg_chord):
    return re.sub('[#bx]', '', rearrg_chord)
```

7.6 Get the roman chord number and find inversion

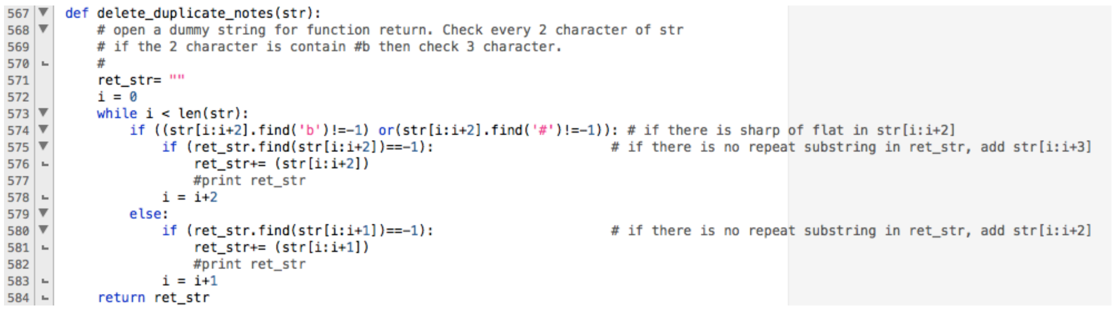
We can check the inversion by getting the original input chord and remove the sharp/flat. Then, we will know which note will become the very lowest one. Here we use a library called music21, which is imported in the beginning. There is a class called music21.pitch.Pitch() . Each note object has a Pitch object embedded into it. There are many method to find the characteristic of a note such as note name, note octave, note step etc. Here, we use Pitch.ps to find the number that representing a note. For example, number 45 can represent "A2" note. Number 60 can represent "C4" note. This is a good way to compare different note in music.



```
def check_inversion(original_input_chord): #matched_possible_chord eg. FAC, ACE, no sharp fla
    #do simplify here to remove sharp flat
    simplified_ori_chord= re.sub('[!@#$b]', '', original_input_chord)
    #check the 0-1,2-3,4-5 pitch.ps of simplified_original eg F2A3C2
    position_0_1 = music21.pitch.Pitch(simplified_ori_chord[:2])
    position_2_3 =music21.pitch.Pitch(simplified_ori_chord[2:-2])
    position_4_5 = music21.pitch.Pitch(simplified_ori_chord[4:])

    # 0_1<2_3<4_5 // 2_3<4_5<0_1 // 4_5<0_1<2_3
    if((position_0_1.ps<position_2_3.ps)&(position_0_1.ps<position_4_5.ps)):
        smallest = "root"
    elif((position_2_3.ps<position_4_5.ps)&(position_2_3.ps<position_0_1.ps)):
        smallest = "63"
    elif((position_4_5.ps<position_0_1.ps)&(position_4_5.ps<position_2_3.ps)):
        smallest = "64"
    else:
        smallest = ""
    return smallest
```

7.7 Delete duplicate notes

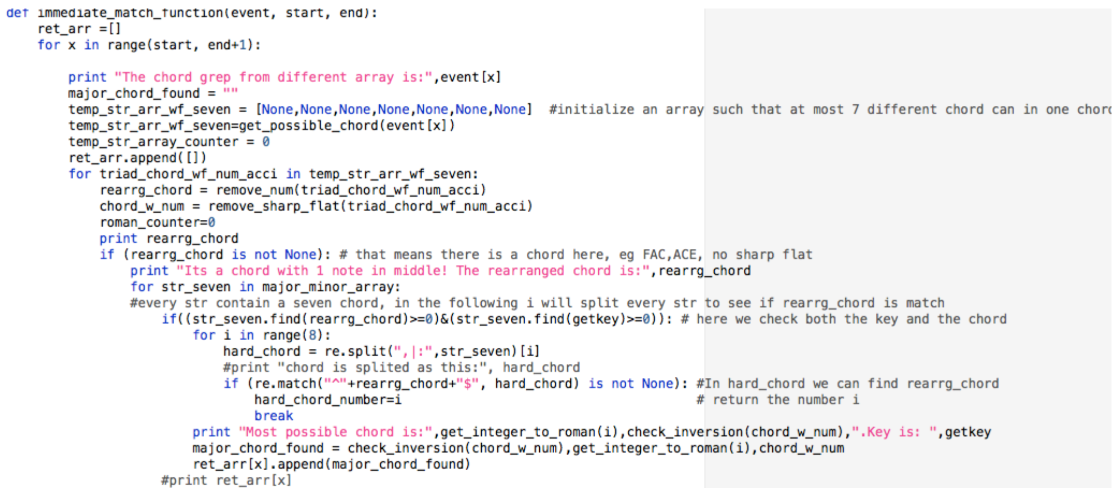
When we get notes from different parts, we may encounter some duplicate notes. Since we want to enhance the speed of the identification of chords, we delete the duplicate ones.



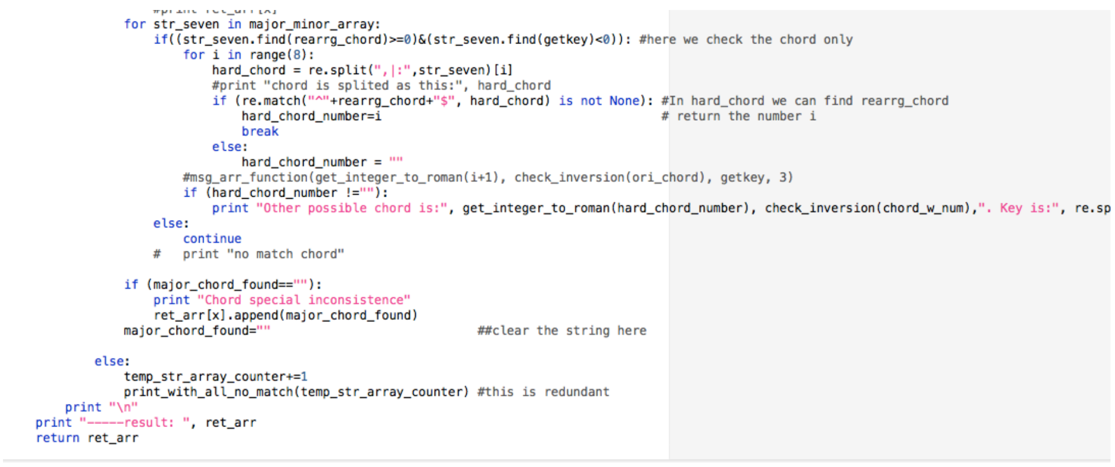
```python
def delete_duplicate_notes(str):
    # open a dummy string for function return. Check every 2 character of str
    # if the 2 character is contain #b then check 3 character.
    #
    ret_str= ""
    i = 0
    while i < len(str):
        if ((str[i:i+2].find('b')!=-1) or(str[i:i+2].find('#')!=-1)): # if there is sharp of flat in str[i:i+2]
            if (ret_str.find(str[i:i+2])==-1):                      # if there is no repeat substring in ret_str, add str[i:i+3]
                ret_str+= (str[i:i+2])
                #print ret_str
            i = i+2
        else:
            if (ret_str.find(str[i:i+1])==-1):                      # if there is no repeat substring in ret_str, add str[i:i+2]
                ret_str+= (str[i:i+1])
                #print ret_str
            i = i+1
    return ret_str
```

7.8 Immediate match of chords

We use this function to identify chords. Input name event is an array contains all the information needed for identifying chords. For example event[0] = ["ACE"] which is grape from different parts of music score. We make use of the array to do analysis.



```python
def immediate_match_function(event, start, end):
    ret_arr =[]
    for x in range(start, end+1):

        print "The chord grep from different array is:",event[x]
        major_chord_found = ""
        temp_str_arr_wf_seven = [None,None,None,None,None,None,None]  #initialize an array such that at most 7 different chord can in one chord
        temp_str_arr_wf_seven=get_possible_chord(event[x])
        temp_str_array_counter = 0
        ret_arr.append([])
        for triad_chord_wf_num_acci in temp_str_arr_wf_seven:
            rearrg_chord = remove_num(triad_chord_wf_num_acci)
            chord_w_num = remove_sharp_flat(triad_chord_wf_num_acci)
            roman_counter=0
            print rearrg_chord
            if (rearrg_chord is not None): # that means there is a chord here, eg FAC,ACE, no sharp flat
                print "Its a chord with 1 note in middle! The rearranged chord is:",rearrg_chord
                for str_seven in major_minor_array:
                #every str contain a seven chord, in the following i will split every str to see if rearrg_chord is match
                    if((str_seven.find(rearrg_chord)>=0)&(str_seven.find(getkey)>=0)): # here we check both the key and the chord
                        for i in range(8):
                            hard_chord = re.split(",|:",str_seven)[i]
                            #print "chord is splited as this:", hard_chord
                            if (re.match("^"+rearrg_chord+"$", hard_chord) is not None): #In hard_chord we can find rearrg_chord
                                hard_chord_number=i                                      # return the number i
                                break
                        print "Most possible chord is:",get_integer_to_roman(i),check_inversion(chord_w_num),".Key is: ",getkey
                        major_chord_found = check_inversion(chord_w_num),get_integer_to_roman(i),chord_w_num
                        ret_arr[x].append(major_chord_found)
                    #print ret_arr[x]
```



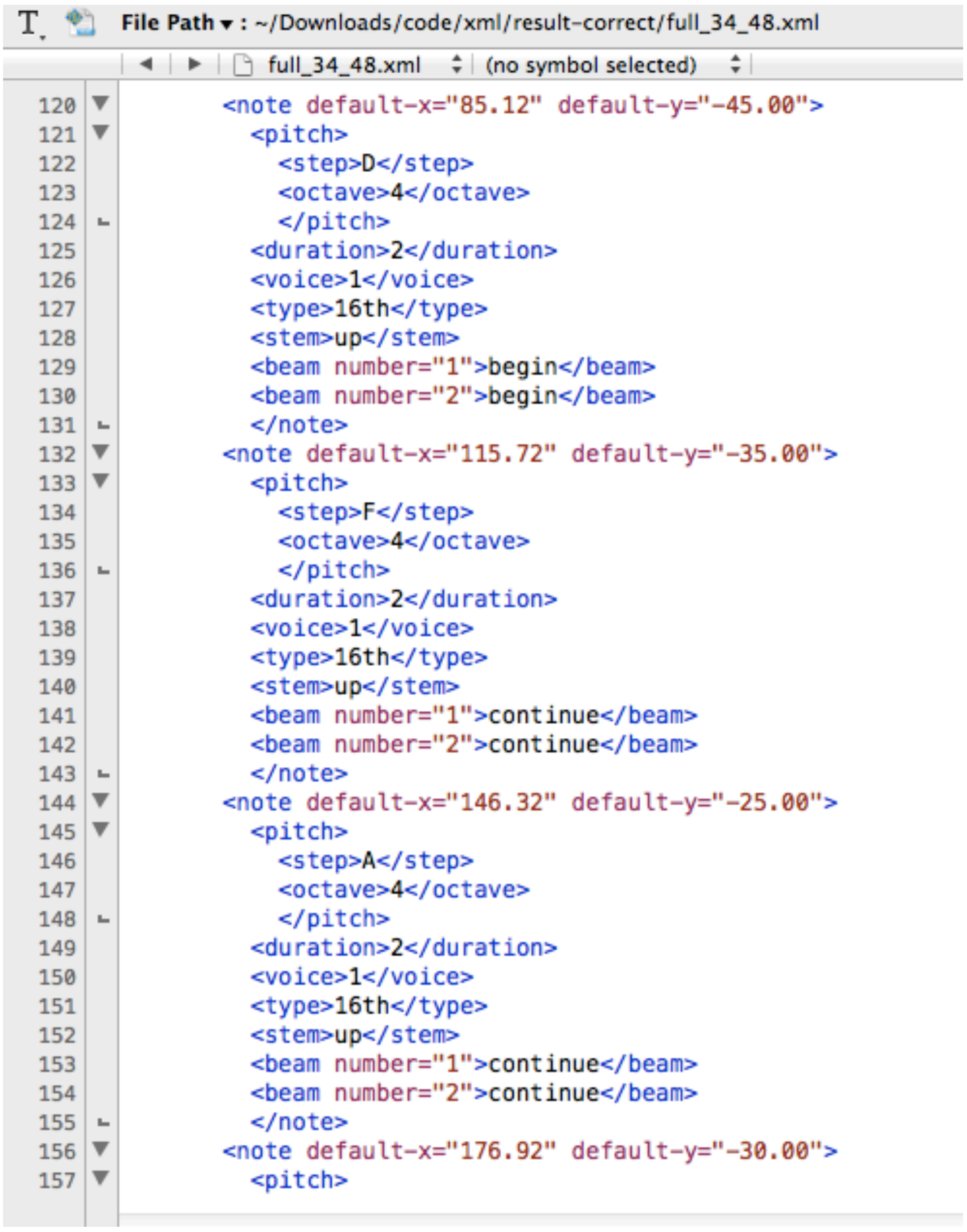
```
            for str_seven in major_minor_array:
                if((str_seven.find(rearrg_chord)>=0)&(str_seven.find(getkey)<0)): #here we check the chord only
                    for i in range(8):
                        hard_chord = re.split(",|:",str_seven)[i]
                        #print "chord is splited as this:", hard_chord
                        if (re.match("^"+rearrg_chord+"$", hard_chord) is not None): #In hard_chord we can find rearrg_chord
                            hard_chord_number=i                                      # return the number i
                            break
                        else:
                            hard_chord_number = ""
                    #msg_arr_function(get_integer_to_roman(i+1), check_inversion(ori_chord), getkey, 3)
                    if (hard_chord_number !=""):
                        print "Other possible chord is:", get_integer_to_roman(hard_chord_number), check_inversion(chord_w_num),". Key is:", re.sp
                else:
                    continue
                #   print "no match chord"

            if (major_chord_found==""):
                print "Chord special inconsistence"
                ret_arr[x].append(major_chord_found)
            major_chord_found=""                          ##clear the string here

        else:
            temp_str_array_counter+=1
            print_with_all_no_match(temp_str_array_counter) #this is redundant
    print "\n"
print "-----result: ", ret_arr
return ret_arr
```

7.9 Input file using music21

The original file containing all the notes is in XML format. In other words, tags structure the whole music score. For example, if there is a note with pitch "D" and duration is 2, then it will formatted as:

<pitch><step>D</step></pitch><duration>2</duration>.

If we read the file using normal input/output stream provide in python, the chance of getting error raise. It is because we first need to found out the tags with lots of if-else clause, it is not easy to debug. Therefore, we install Music21 module to work with Python in Mac.



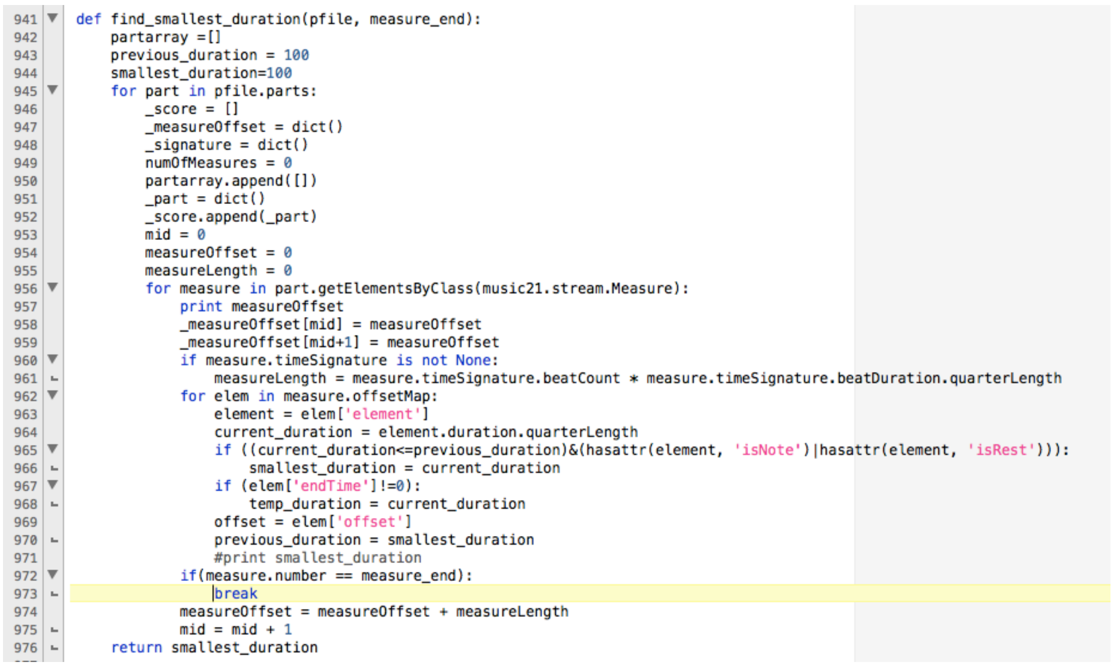
File Path ▾ : ~/Downloads/code/xml/result-correct/full_34_48.xml

full_34_48.xml (no symbol selected)

```
        <note default-x="85.12" default-y="-45.00">
          <pitch>
            <step>D</step>
            <octave>4</octave>
            </pitch>
          <duration>2</duration>
          <voice>1</voice>
          <type>16th</type>
          <stem>up</stem>
          <beam number="1">begin</beam>
          <beam number="2">begin</beam>
          </note>
        <note default-x="115.72" default-y="-35.00">
          <pitch>
            <step>F</step>
            <octave>4</octave>
            </pitch>
          <duration>2</duration>
          <voice>1</voice>
          <type>16th</type>
          <stem>up</stem>
          <beam number="1">continue</beam>
          <beam number="2">continue</beam>
          </note>
        <note default-x="146.32" default-y="-25.00">
          <pitch>
            <step>A</step>
            <octave>4</octave>
            </pitch>
          <duration>2</duration>
          <voice>1</voice>
          <type>16th</type>
          <stem>up</stem>
          <beam number="1">continue</beam>
          <beam number="2">continue</beam>
          </note>
        <note default-x="176.92" default-y="-30.00">
          <pitch>
```

7.10 For finding a shortest duration note in particular measure

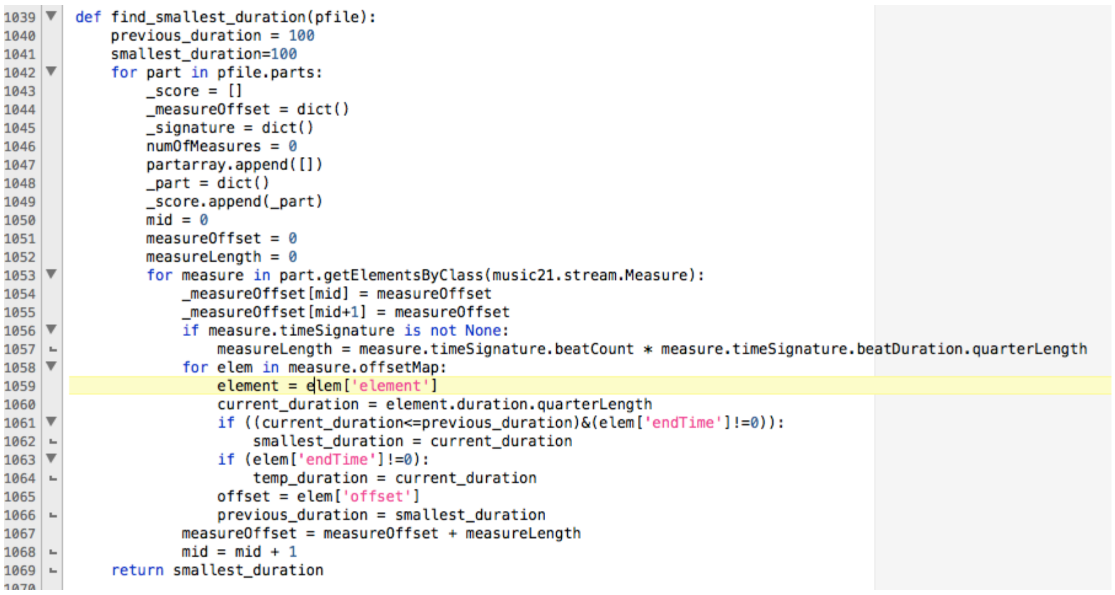
The function has two attributes. First attribute is the file, the other is when the measure ends. We don't want to scan the whole score, since the whole score may contain some note that have too short duration. It is very clumsy for us to do chord identifying, if we divide the note into too short duration.



```python
def find_smallest_duration(pfile, measure_end):
    partarray =[]
    previous_duration = 100
    smallest_duration=100
    for part in pfile.parts:
        _score = []
        _measureOffset = dict()
        _signature = dict()
        numOfMeasures = 0
        partarray.append([])
        _part = dict()
        _score.append(_part)
        mid = 0
        measureOffset = 0
        measureLength = 0
        for measure in part.getElementsByClass(music21.stream.Measure):
            print measureOffset
            _measureOffset[mid] = measureOffset
            _measureOffset[mid+1] = measureOffset
            if measure.timeSignature is not None:
                measureLength = measure.timeSignature.beatCount * measure.timeSignature.beatDuration.quarterLength
            for elem in measure.offsetMap:
                element = elem['element']
                current_duration = element.duration.quarterLength
                if ((current_duration<=previous_duration)&(hasattr(element, 'isNote')|hasattr(element, 'isRest'))):
                    smallest_duration = current_duration
                if (elem['endTime']!=0):
                    temp_duration = current_duration
                offset = elem['offset']
                previous_duration = smallest_duration
                #print smallest_duration
            if(measure.number == measure_end):
                break
            measureOffset = measureOffset + measureLength
            mid = mid + 1
    return smallest_duration
```

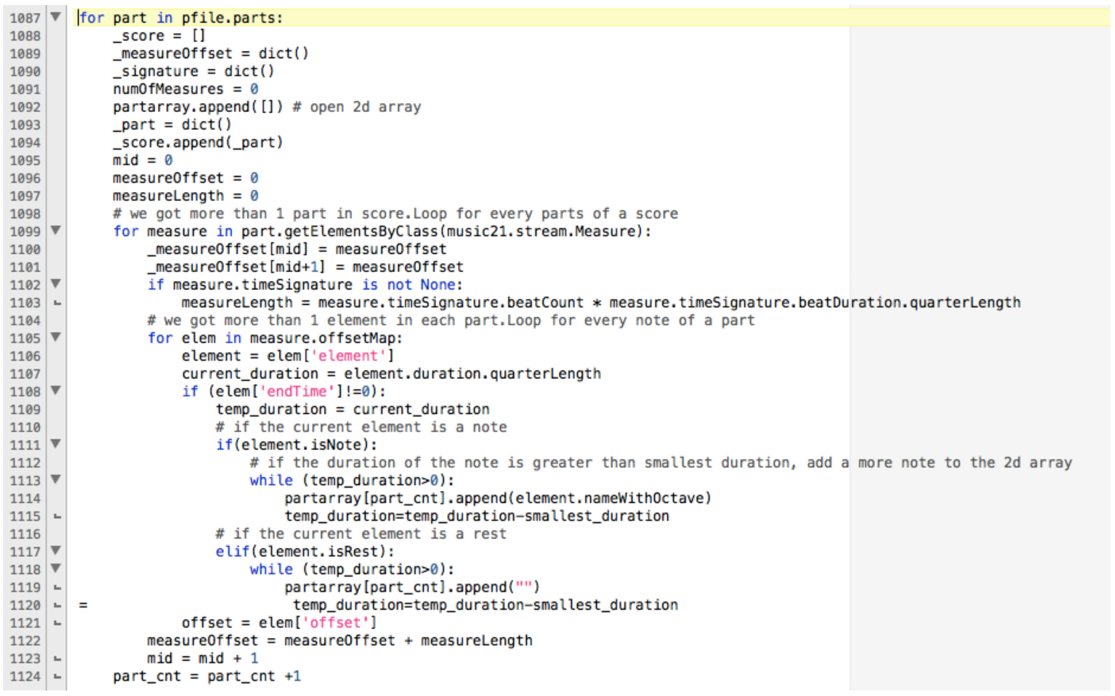
7.11 For changing a long duration note into several short duration notes

Since we need to change the long duration note into a note with short duration. We will need to find out the shortest duration. The function below finds out the smallest duration in the score.



```python
def find_smallest_duration(pfile):
    previous_duration = 100
    smallest_duration=100
    for part in pfile.parts:
        _score = []
        _measureOffset = dict()
        _signature = dict()
        numOfMeasures = 0
        partarray.append([])
        _part = dict()
        _score.append(_part)
        mid = 0
        measureOffset = 0
        measureLength = 0
        for measure in part.getElementsByClass(music21.stream.Measure):
            _measureOffset[mid] = measureOffset
            _measureOffset[mid+1] = measureOffset
            if measure.timeSignature is not None:
                measureLength = measure.timeSignature.beatCount * measure.timeSignature.beatDuration.quarterLength
            for elem in measure.offsetMap:
                element = elem['element']
                current_duration = element.duration.quarterLength
                if ((current_duration<=previous_duration)&(elem['endTime']!=0)):
                    smallest_duration = current_duration
                if (elem['endTime']!=0):
                    temp_duration = current_duration
                offset = elem['offset']
                previous_duration = smallest_duration
            measureOffset = measureOffset + measureLength
            mid = mid + 1
    return smallest_duration
```

The function below makes use of module from Music21. We first generate a for-loop to check each part of music score. Then for each part of music score, we generate a for-loop to check each element. If the element is a note, we check whether duration of element is longer than the shortest duration. If not, we put the element into an array and look for the next element. If yes, we put the element into an array, and do subtraction. Then look for the same element until the element's duration is shorter than shortest duration.



```python
for part in pfile.parts:
    _score = []
    _measureOffset = dict()
    _signature = dict()
    numOfMeasures = 0
    partarray.append([]) # open 2d array
    _part = dict()
    _score.append(_part)
    mid = 0
    measureOffset = 0
    measureLength = 0
    # we got more than 1 part in score.Loop for every parts of a score
    for measure in part.getElementsByClass(music21.stream.Measure):
        _measureOffset[mid] = measureOffset
        _measureOffset[mid+1] = measureOffset
        if measure.timeSignature is not None:
            measureLength = measure.timeSignature.beatCount * measure.timeSignature.beatDuration.quarterLength
        # we got more than 1 element in each part.Loop for every note of a part
        for elem in measure.offsetMap:
            element = elem['element']
            current_duration = element.duration.quarterLength
            if (elem['endTime']!=0):
                temp_duration = current_duration
                # if the current element is a note
                if(element.isNote):
                    # if the duration of the note is greater than smallest duration, add a more note to the 2d array
                    while (temp_duration>0):
                        partarray[part_cnt].append(element.nameWithOctave)
                        temp_duration=temp_duration-smallest_duration
                # if the current element is a rest
                elif(element.isRest):
                    while (temp_duration>0):
                        partarray[part_cnt].append("")
=                         temp_duration=temp_duration-smallest_duration
            offset = elem['offset']
        measureOffset = measureOffset + measureLength
        mid = mid + 1
    part_cnt = part_cnt +1
```

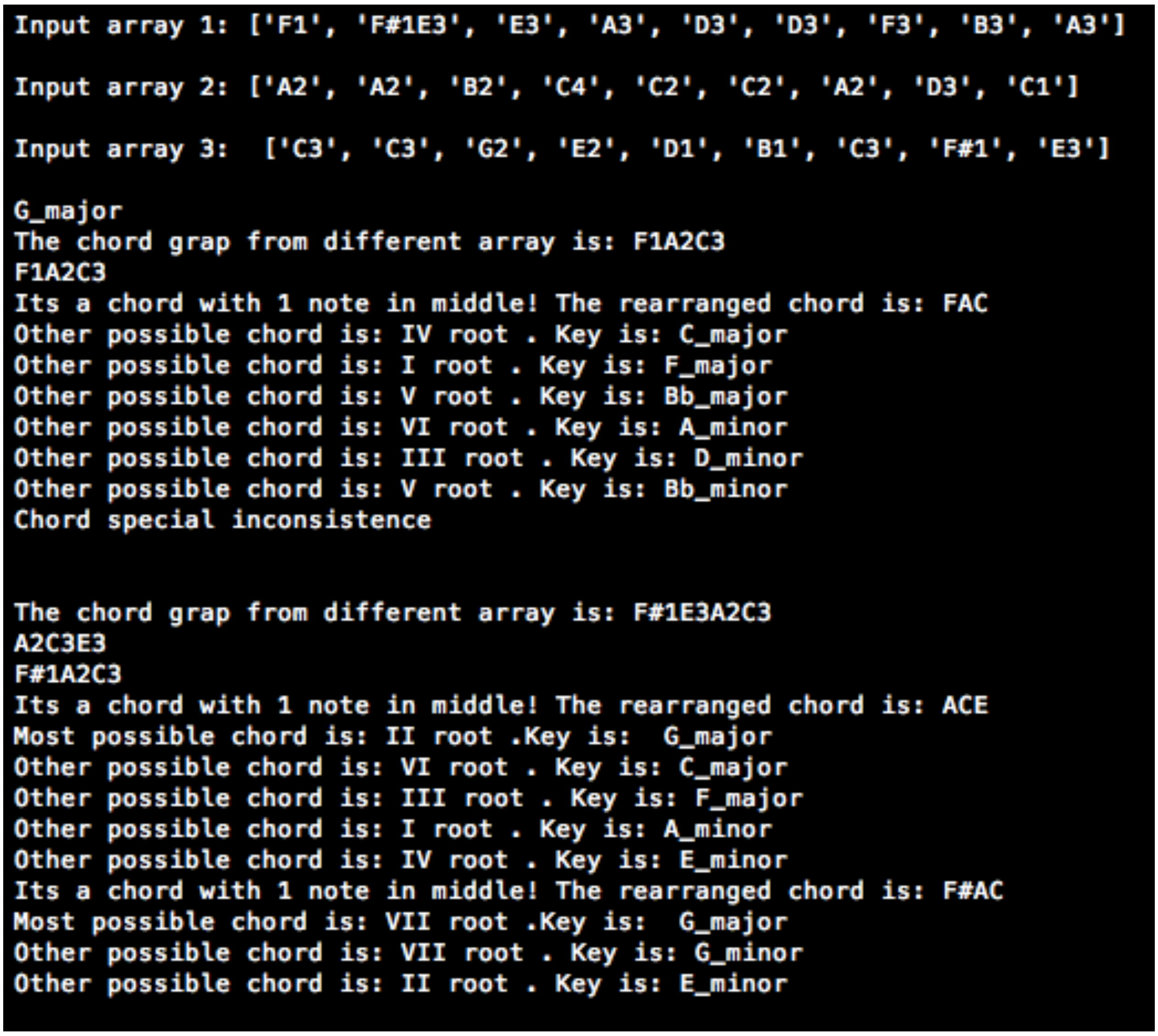
8. Results

8.1 Result for last semester

First of all, we assume it is a G major scale.

For the first chord, we assume to have F1A2C3. Then we can rearrange the chord by removing the octave and change it into FAC. After that it can be used for matching with the database. Since the database got no match with a G major and a chord with FAC. Therefore, there is no most possible case in the this case. However, we still got other possibilities in other major/minor scale. For example, we can still found that a chord in C major and the chord is IV root.

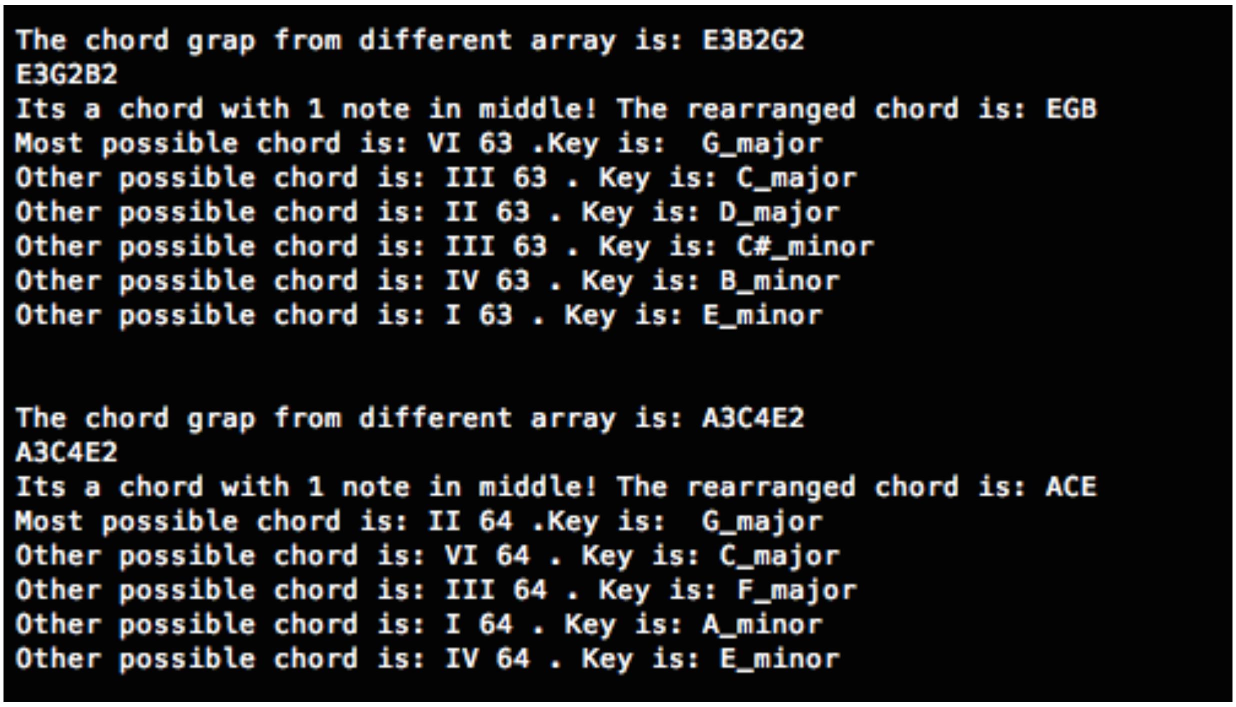
For the second chord, we have F#1E3A2C3. Then we can rearrange the chord by removing the octave and change it into ACE and F#AC. For chord ACE, there is a most possible chord match correctly with database. And for chord F#AC, it is also a VII chord in G major.



```
Input array 1: ['F1', 'F#1E3', 'E3', 'A3', 'D3', 'D3', 'F3', 'B3', 'A3']

Input array 2: ['A2', 'A2', 'B2', 'C4', 'C2', 'C2', 'A2', 'D3', 'C1']

Input array 3:  ['C3', 'C3', 'G2', 'E2', 'D1', 'B1', 'C3', 'F#1', 'E3']

G_major
The chord grap from different array is: F1A2C3
F1A2C3
Its a chord with 1 note in middle! The rearranged chord is: FAC
Other possible chord is: IV root . Key is: C_major
Other possible chord is: I root . Key is: F_major
Other possible chord is: V root . Key is: Bb_major
Other possible chord is: VI root . Key is: A_minor
Other possible chord is: III root . Key is: D_minor
Other possible chord is: V root . Key is: Bb_minor
Chord special inconsistence


The chord grap from different array is: F#1E3A2C3
A2C3E3
F#1A2C3
Its a chord with 1 note in middle! The rearranged chord is: ACE
Most possible chord is: II root .Key is:  G_major
Other possible chord is: VI root . Key is: C_major
Other possible chord is: III root . Key is: F_major
Other possible chord is: I root . Key is: A_minor
Other possible chord is: IV root . Key is: E_minor
Its a chord with 1 note in middle! The rearranged chord is: F#AC
Most possible chord is: VII root .Key is:  G_major
Other possible chord is: VII root . Key is: G_minor
Other possible chord is: II root . Key is: E_minor
```

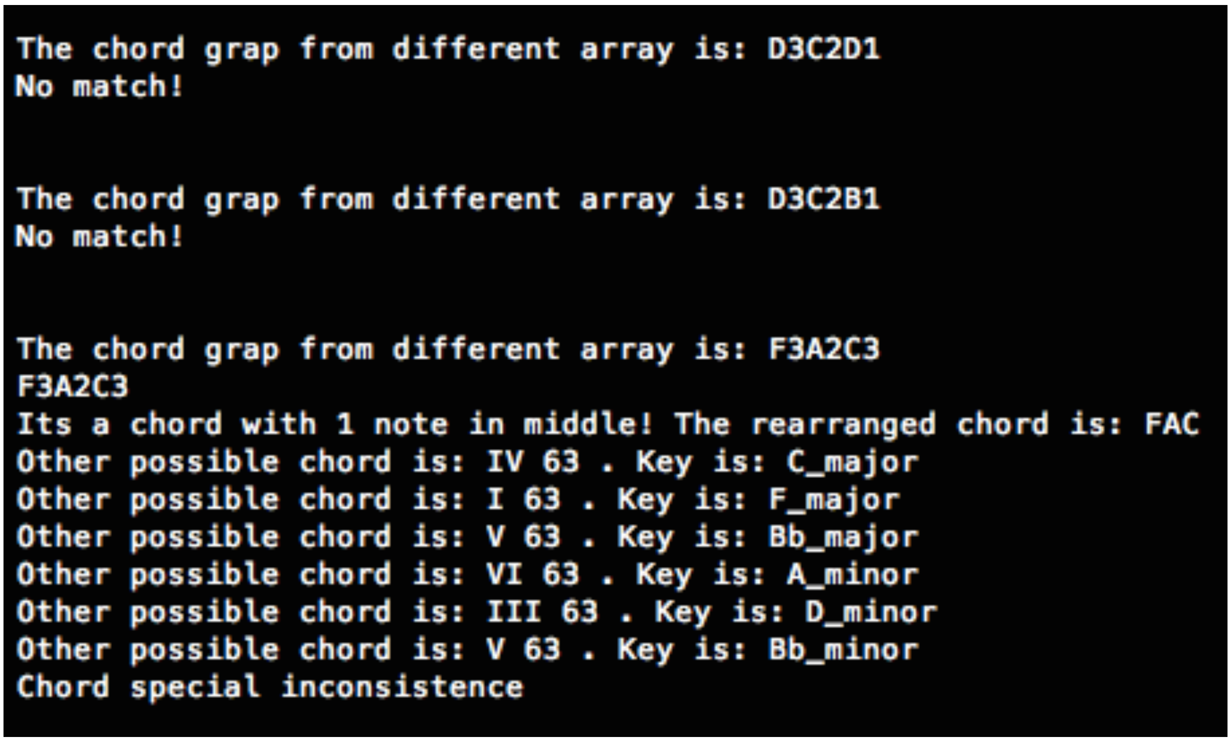
For the third chord E3B2G2, we got an inversion here. Since if we rearrange the chord from the lowest pitch to highest pitch, we have B2G2E3. The note B is at the bottom such that it is a first inversion and we can say it is VI63.

For the fourth chord A3C4E2, we got an inversion here. Since if we rearrange the chord from the lowest pitch to highest pitch, we have E2A3C4. The note E is at the bottom such that it is a second inversion and we can say it is VI64.


```
The chord grap from different array is: E3B2G2
E3G2B2
Its a chord with 1 note in middle! The rearranged chord is: EGB
Most possible chord is: VI 63 .Key is:  G_major
Other possible chord is: III 63 . Key is: C_major
Other possible chord is: II 63 . Key is: D_major
Other possible chord is: III 63 . Key is: C#_minor
Other possible chord is: IV 63 . Key is: B_minor
Other possible chord is: I 63 . Key is: E_minor


The chord grap from different array is: A3C4E2
A3C4E2
Its a chord with 1 note in middle! The rearranged chord is: ACE
Most possible chord is: II 64 .Key is:  G_major
Other possible chord is: VI 64 . Key is: C_major
Other possible chord is: III 64 . Key is: F_major
Other possible chord is: I 64 . Key is: A_minor
Other possible chord is: IV 64 . Key is: E_minor
```

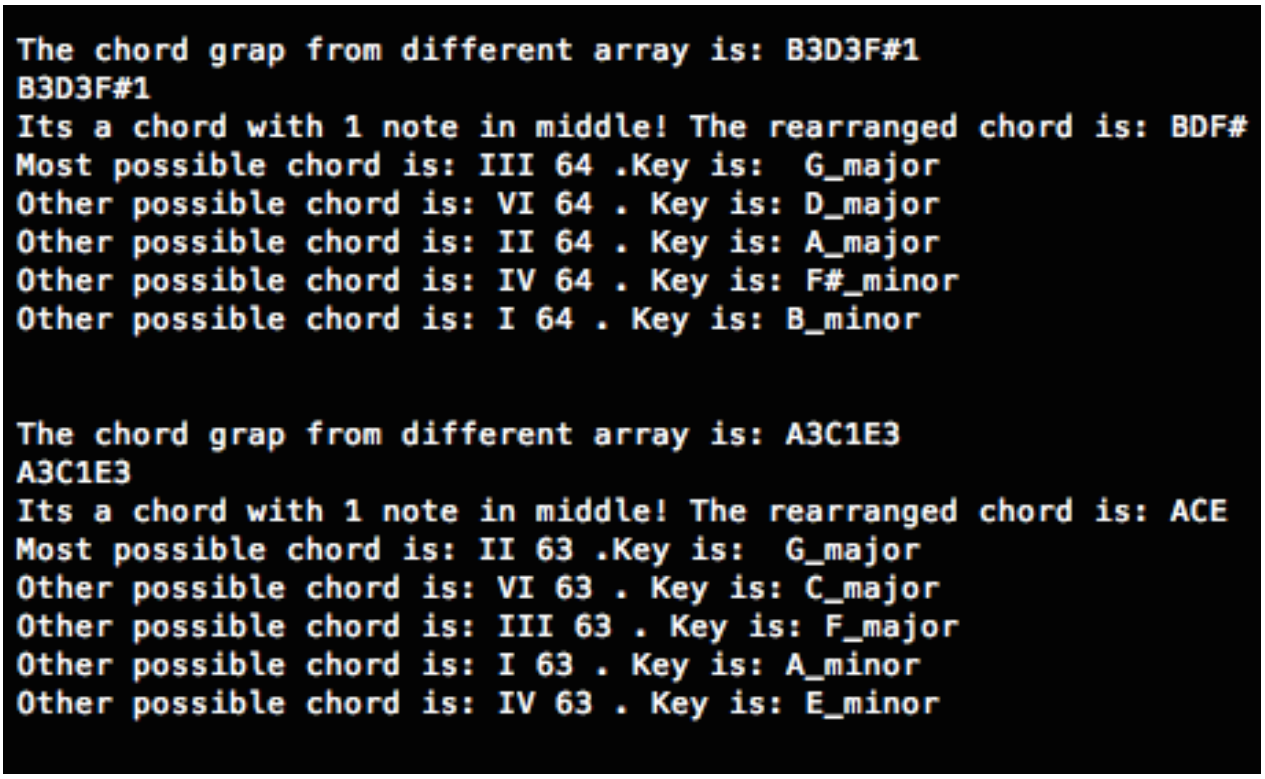
For fifth and sixth case, which are D3C2D1 and D3C2B1 respectively, after we rearrange the chord, we cannot find any chord that contains DCD or DCB in any major/minor. So the terminal output a message "no match" for these cases.

For seventh cases F2A2C3, since the database got no match with a G major and a chord with FAC. Therefore, there is no most possible case in the this case. However, we still got other possibilities in other major/minor scale.



```
The chord grap from different array is: D3C2D1
No match!


The chord grap from different array is: D3C2B1
No match!


The chord grap from different array is: F3A2C3
F3A2C3
Its a chord with 1 note in middle! The rearranged chord is: FAC
Other possible chord is: IV 63 . Key is: C_major
Other possible chord is: I 63 . Key is: F_major
Other possible chord is: V 63 . Key is: Bb_major
Other possible chord is: VI 63 . Key is: A_minor
Other possible chord is: III 63 . Key is: D_minor
Other possible chord is: V 63 . Key is: Bb_minor
Chord special inconsistence
```

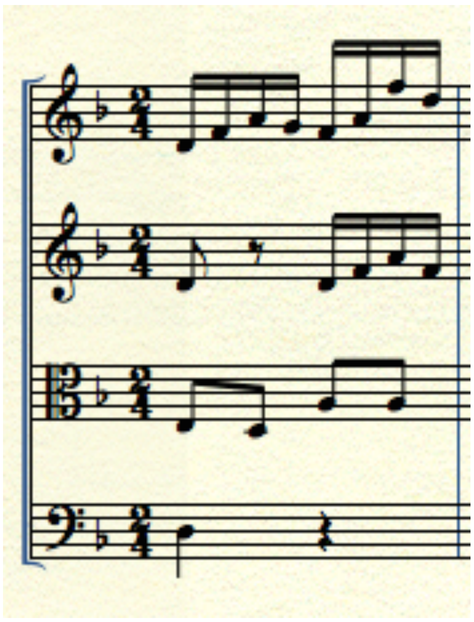
For seventh cases B3D3F#1 and eighth case, we got the corresponding chord and inversion here.



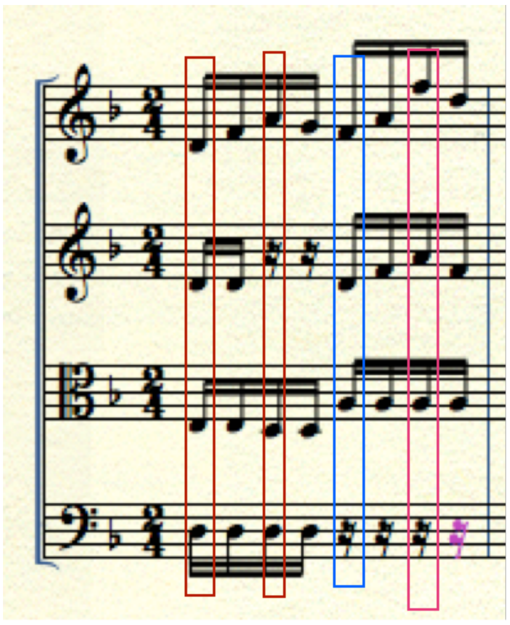
```
The chord grap from different array is: B3D3F#1
B3D3F#1
Its a chord with 1 note in middle! The rearranged chord is: BDF#
Most possible chord is: III 64 .Key is:  G_major
Other possible chord is: VI 64 . Key is: D_major
Other possible chord is: II 64 . Key is: A_major
Other possible chord is: IV 64 . Key is: F#_minor
Other possible chord is: I 64 . Key is: B_minor


The chord grap from different array is: A3C1E3
A3C1E3
Its a chord with 1 note in middle! The rearranged chord is: ACE
Most possible chord is: II 63 .Key is:  G_major
Other possible chord is: VI 63 . Key is: C_major
Other possible chord is: III 63 . Key is: F_major
Other possible chord is: I 63 . Key is: A_minor
Other possible chord is: IV 63 . Key is: E_minor
```

8.2 Result for this semester

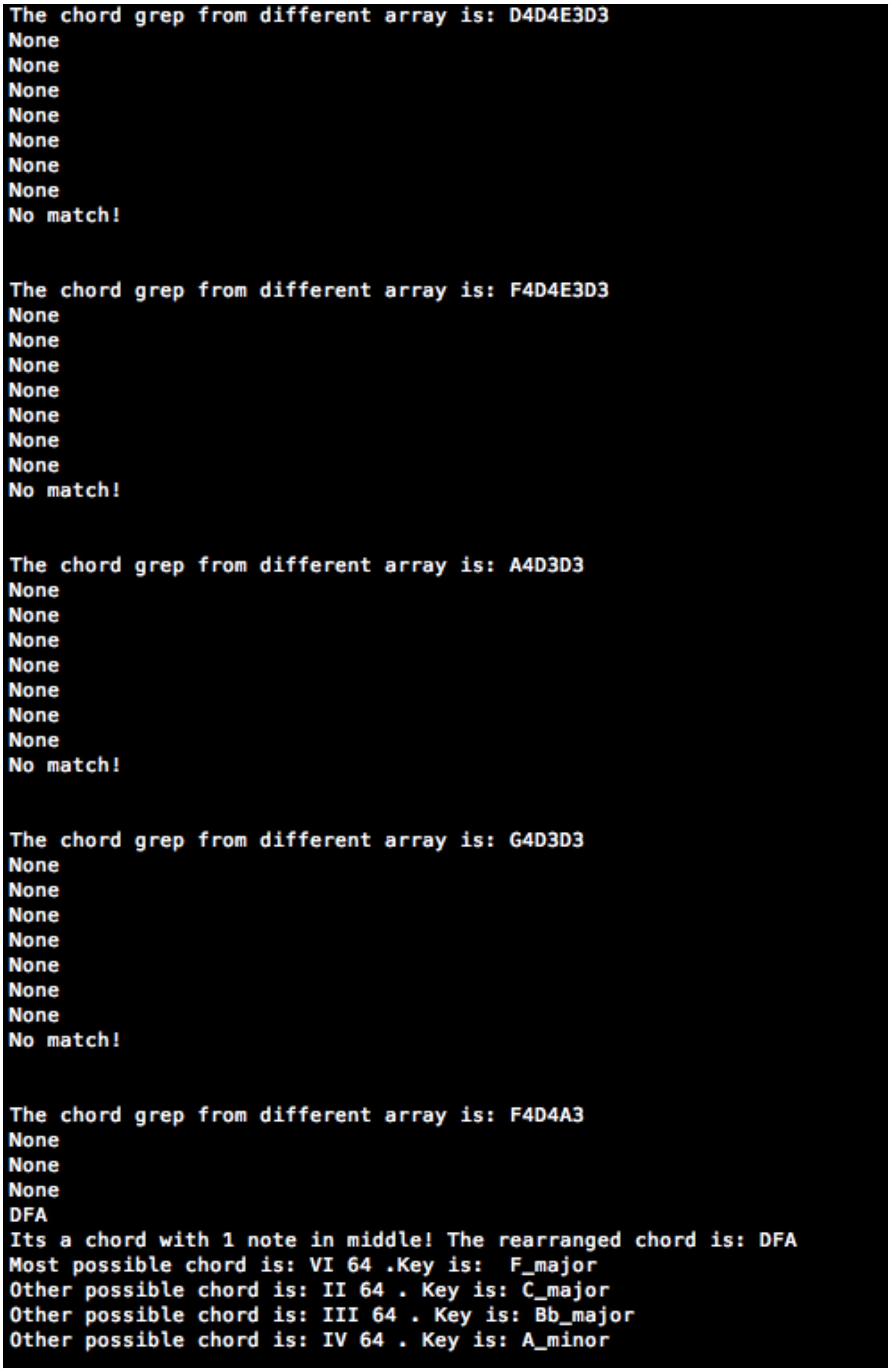
In this semester, we will use music21 to do input of file. Below is one of the files.



We will do analysis vertically just like the figure below. Then we can see there is a chord. The rectangle in blue color indicates the chord "DFA".

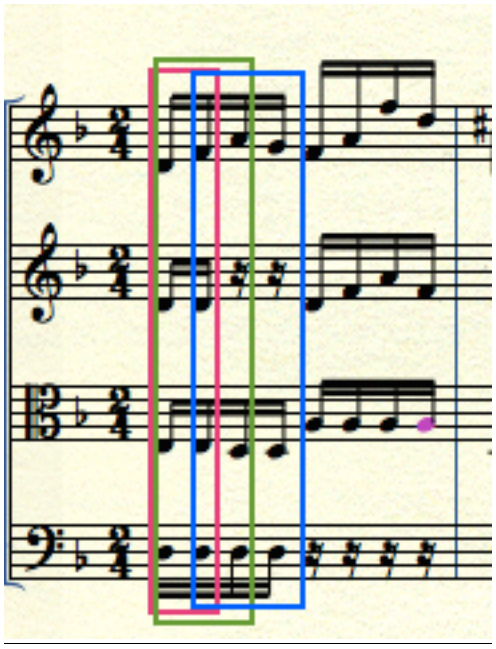


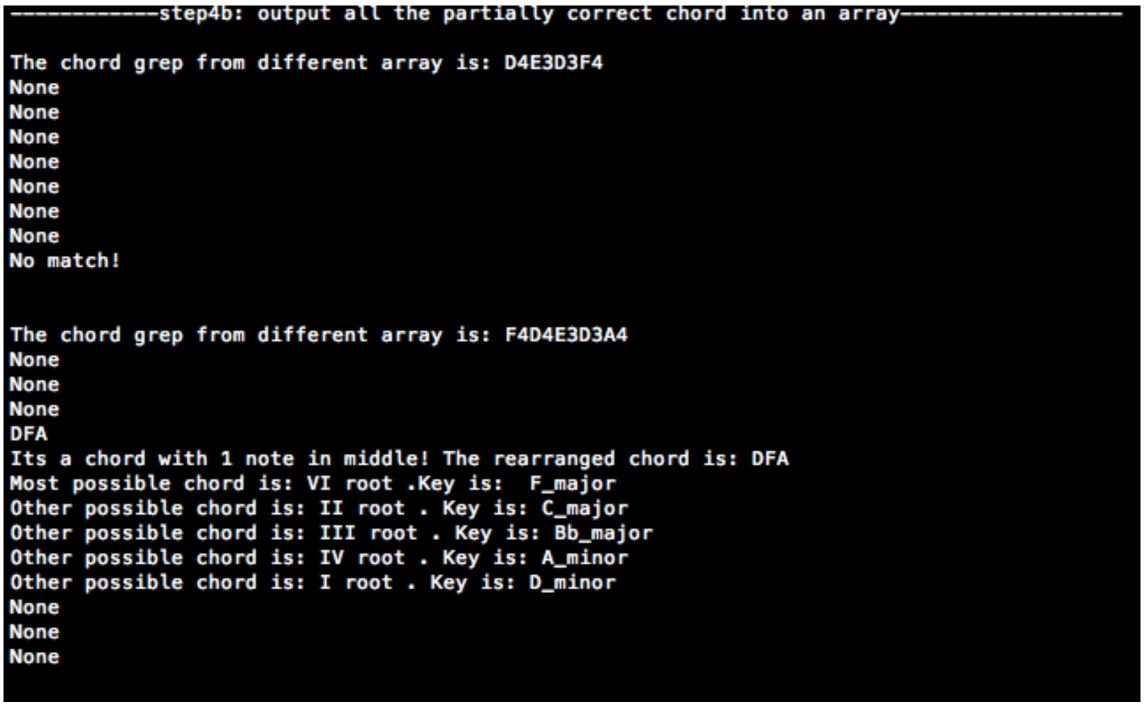
The figure below shows the result. The program can find out the pitches in blue rectangle are “D4”-”F4”-”A3”. Therefore, it can determine whether it is a chord by using algorithm developed. Once it finds out the chord, it will immediately find out whether there is inversion and then output the inversion found.


```
The chord grep from different array is: D4D4E3D3
None
None
None
None
None
None
None
No match!


The chord grep from different array is: F4D4E3D3
None
None
None
None
None
None
None
No match!


The chord grep from different array is: A4D3D3
None
None
None
None
None
None
None
No match!


The chord grep from different array is: G4D3D3
None
None
None
None
None
None
None
No match!


The chord grep from different array is: F4D4A3
None
None
None
DFA
Its a chord with 1 note in middle! The rearranged chord is: DFA
Most possible chord is: VI 64 .Key is:  F_major
Other possible chord is: II 64 . Key is: C_major
Other possible chord is: III 64 . Key is: Bb_major
Other possible chord is: IV 64 . Key is: A_minor
```

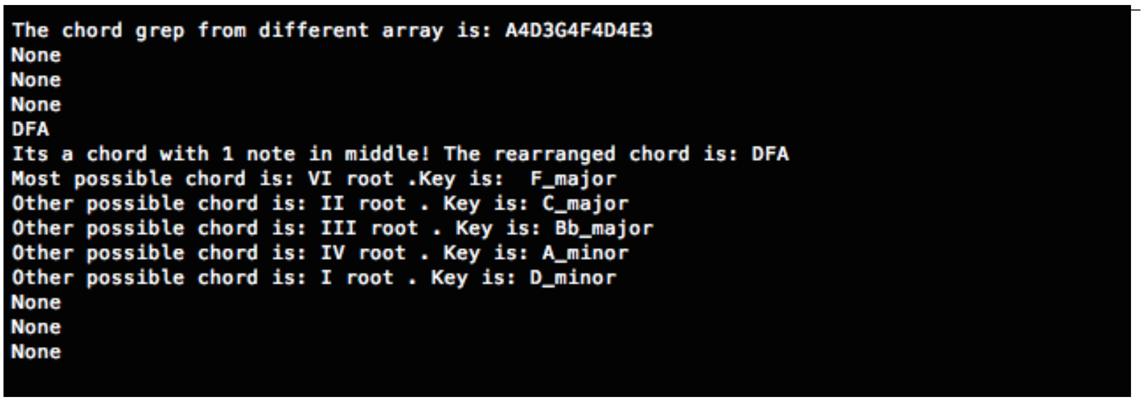
However, some results still contain no chord. For example, we cannot get any result in the first four analyses. Then we will combine the 3 pitch class grape with previous and the following one.

We can see the figure below. We combine the first one with the second (red rectangle). Then we store the combined pitches in am array. After that, we combine the second with the previous one and the following one (green rectangle). Then we do the analyses again.





```
------------step4b: output all the partially correct chord into an array------------------

The chord grep from different array is: D4E3D3F4
None
None
None
None
None
None
None
No match!


The chord grep from different array is: F4D4E3D3A4
None
None
None
DFA
Its a chord with 1 note in middle! The rearranged chord is: DFA
Most possible chord is: VI root .Key is:  F_major
Other possible chord is: II root . Key is: C_major
Other possible chord is: III root . Key is: Bb_major
Other possible chord is: IV root . Key is: A_minor
Other possible chord is: I root . Key is: D_minor
None
None
None
```



```
The chord grep from different array is: A4D3G4F4D4E3
None
None
None
DFA
Its a chord with 1 note in middle! The rearranged chord is: DFA
Most possible chord is: VI root .Key is:  F_major
Other possible chord is: II root . Key is: C_major
Other possible chord is: III root . Key is: Bb_major
Other possible chord is: IV root . Key is: A_minor
Other possible chord is: I root . Key is: D_minor
None
None
None
```

9. Conclusion

To conclude, it is all known that chord is important in music. It gives a good opportunity for instrument to make the whole piece of music become more harmony. In our objective, we have to find the chord in the music xml and define it programmatically.

Our first step is to familiarize with basic key, notes and major/minor definition. As for music, those basic elements are crucial in composing music scores. It can generate a melody in either sad or happy way.

Our second step is to familiarize with Music XML and its parser. Since Music XML file is a bunch of tags and attributes. If we directly edit the XML file, errors may easily occur. Therefore, we need a Python and a library called Music21. We want to find if there is any useful function in Music21 for finding a chord. However, there is no useful function. Some function in Music21 may contain similar property in finding chords. But due to its low accuracy, we decided to develop our own program to find the chords.

The difficulties we solved:

- Since the XML file need to be read by Music21's module in order to reduce the chance of getting error. We have to study the Music21's module, which is used in inputting XML files.
- We do not scan the whole score for the shortest duration since it is not favorable for identify chords. Therefore, we stop scanning notes when a particular measure is reach
- At the beginning we do not know whether store the notes horizontally one by one or store the notes vertically one by one. At last, we choose to store the whole score horizontally first. And then do the analysis vertically.

The difficulties we encounter:

- There is no function in collecting chords. We need to collect chords by opening some arrays and extract chords from several arrays.
- The Music21 module is brand new to us. We need to spend time to find out any usable module to use.
- There modulation confirms may occur after a pivot is found. We need to collect chords after several measures.

Thing we would like to improve:

- To combine with the project done last year
- Output a XML file

10. References

10.1 Acknowledgements

This project was co-supervised by Professor Lucas Wong, who is a professor in Music. He is able to show how to identify chords and shows how a modulation or borrowed occur. This project cannot be done without his assistance.

10.2 Open-source Libraries and book reference

This project is implemeted with the help of several open-source libraries:

`http://www.musicxml.com`

`http://web.mit.edu/`

`http://pybrain.org`

`http://www.numpy.org`

Also, there is some book for reference in learning chords.

Music theory fundamentals, high-yield music theory, vol.1

The effect of the major and minor mode in music as a mood induction procedure by D.michelle Hinn http://music.tutsplus.com/tutorials/using-triads-for-melodies-arpeggios-and-rhythm-guitar-part-1--audio-18123

Furthermore, some material is inheritance from Semester 1.