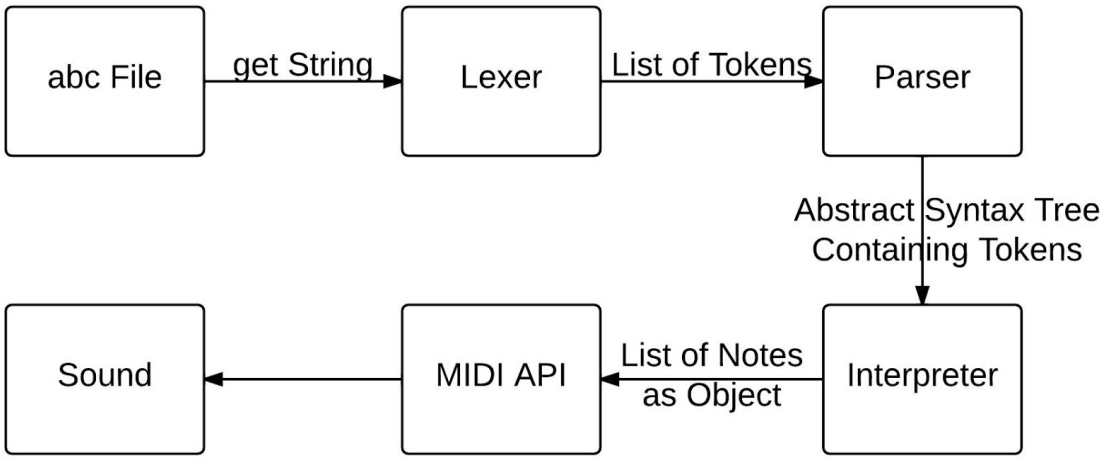
ABC Player

Author: Yonglin Wu, Yang Chen, Menghsuan Pan

Introduction

ABC Player is a Java application that plays an abc file. It takes one abc file at a time, parse it and feed it into Java MIDI API. It has three major parts: lexer, parser and interpreter. The function of each part is explained in details in this document.

Flow Chart:



Lexer

We read the input file line by line and convert each line into a string, Lexer will convert all the strings together into a header(a type defined in header.java) and a list of tokens.

Header

The header has the following fields: C,K,L,M,Q,T,X and it contains information about the whole piece, namely, composer(C), key(K), default length of a note(L), meter(sums of durations of all notes within a bar), Q(tempo), T(title of the piece) and X(index number).

Token

Tokens are generated from characters or substrings from the list. If we see a matching

A token has the following fields:
Type. It specifies what type of token it is. It is an enum type, which is listed below with each type's corresponding string:

LINE: |
ACCIDENTAL: ^ or _ or =
NOTE: a-gA-G
OCTAVE: ' or ,
LEFTBRA: [
RIGHTBRA:]
DUPLET: (2
TRIPLET: (3
QUADRUPLET: (4
COLON: :
Rest: zZ
LENGTH: number that represent the length of the note, such as 2, 1/3.
VOICE: v (followed by the name of the voice)
ALTONE: [1
ALTTWO: [2
SPACE: " "

String the original string value of the token is stored in this field.
Val: the value of the Token. This value is a double, and only applies for Token Length. For other tokens Val = null.

Semantic Checking and Exception Thrown

In general we don't handle grammar error (such as "((" or "^^") in Lexer except for the case of alternative ending in repeats. In Lexer we simply return all the tokens are the parser will throw exception later. We throw an IllegalArgumentException for any of the characters that don't exist in abc grammar, such as '&', '+', etc. We include the whitespaces as tokens.

Something else that is worth noting is that a % symbol will cause the remainder of any input line to be ignored as it will be recognized as comment.

Lexer also throws IllegalArgumetException if the header input has wrong format. It is thrown when the header of the input file doesn't follow the following format:
The first field in the header must be the index number ('X').
The second field in the header must be the title ('T').
The last field in the header must be the key ('K').
Each field in the header occurs on a separate line.

Parser

Parser parses the output of a Lexer (a header and a list of tokens) into an abstract syntax tree. The datatype definition/grammar is shown as below. There are two types of datatypes: terminals, which are the tokens produced from Lexer, and non-

terminals, which are recursively defined by terminals and non-terminals.

Datatype Definition/ Grammar

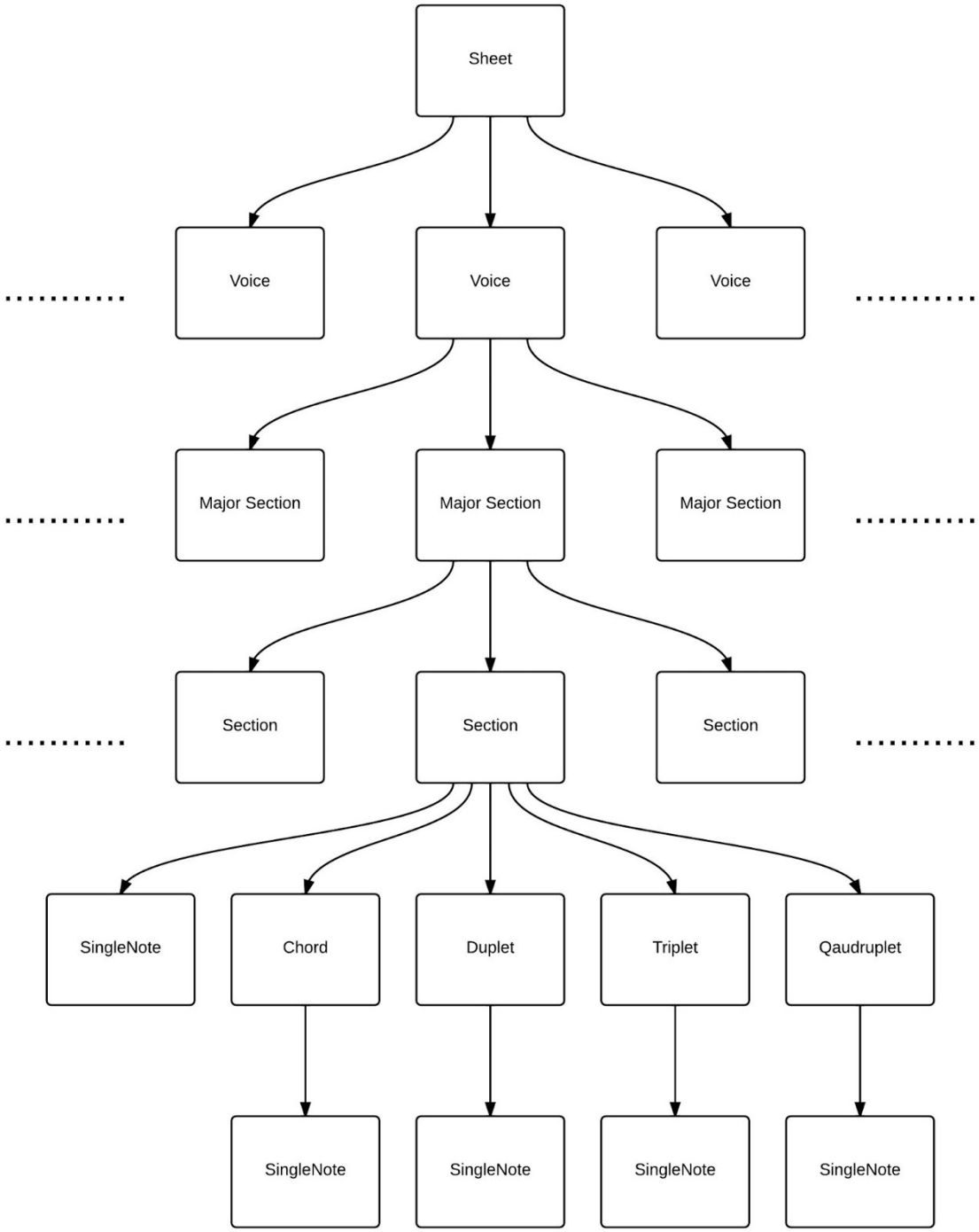
Terminals:

LINE: |
ACCIDENTAL: ^ or _ or =
NOTE: a-gA-G
OCTAVE: ' or ,
LEFTBRA: [
RIGHTBRA:]
DUPLET: (2
TRIPLET: (3
QUADRUPLET: (4
COLON: :
REST: z
LENGTH: number that represent the length of the note, such as 2, 1/3.
VOICE: v: [name of voice]
ALTONE: [1
ALTTWO: [2
SPACE: " "

Non-terminals:

Sheet = Voice*
Voice = MajorSection*
MajorSection = (Section|Repeat)* LINE|RIGHTBRA
Section = Notes LINE
Notes = (SingleNote|Chord|Duplet|Triplet|Quadruplet)*
SingleNote = ACCIDENTAL? [NOTE|REST] OCTAVE* LENGTH?
Chord = LEFTBRA SingleNote+ RIGHTBRA
Duplet = DUPLET SingleNote{2}
Triplet = TRIPLET SingleNote{3}
Quadruplet = QUADRUPLET Single{4}
Rest = [z] Length?
Repeat = COLON? Bar+ ALTONE? Notes COLON LINE (ALTTWO Notes LINE)?

Graph



Semantic Checking and throwing exceptions

If the input file has incorrect syntax, we will throw exceptions. Here are a list of exceptions and when to throw them. All of the exceptions extend RuntimeException class.

IllegalArgumentException:

- SingleNote with more than one length
- SingleNotes that Randomize the following order :

Accidental, Note, Octave, Length

- Duplet that does not have two SingleNote
- Triplet that does not have three SingleNote
- Qaudruplet that does not have four SingleNote
- Chord that includes space

BarLengthException: When the number of notes in a bar is incorrect. (Eg. When the meter is 4/4 and default note length is ¼ and there are only 3 full length note in a bar)

Design Choice:

1. The original design of making a tree with each branch as a different field stored is rejected since cannot easily decide how many branches exist in each Expression (the interface); therefore, choose List as the abstract data type because list is capable of additional Expression when found.
2. MajorSection was not stable when implemented; therefore, have to delete the entire design of MajorSection.
3. When fitting voice with the same voice, we original decide to match two separate lists so that one list store the name while the corresponding list store the value; however, this kind of design will require recursive methods with a possible stack overflow error. Then, HashMap was choose since map is capable of storing name with its value; the capabililty allows an easier way to match the same voices together.
4. When deciding weather directly convert the list of Token in Major to list of SingleNote or create an buffer list of sections, an buffer section is chose since a section has a set number of notes, and the creation of sections can make error searching possible.
5. When deciding weather we want to pass on Duplet, Triplet, and, Qaudruplet, or convert everything to SingleNote to Interpreter, decided to convert everything to SinlgeNote because the interpreter is designed to deal with converting Note to Sound and minimum other thing.
6. For Chord, in the design, it is impossible to pass on as multiple SingleNote because the notes need to be played at the same time; therefore, decide to simply pass on the Chord and let the interpreter do one extra thine: deal with the SingleNote's store in Chord.
7. In treating multiple accidental or octave for a SingleNote, decide that different accidentals or different octaves for a SinlgeNote are valid input for our design because we want to give the composer as much freedom as possible since we can not deny the beauty of art and music no matter what form they are given.

Interpreter

The interpreter (Feeder class) takes a Parser object, traverses through its parsedList of Voice objects, creates a SequencePlayer from the parameters specified in the header, and inputs the translated midi notes into the SequencePlayer.

Design Choice:

1. The major challenge of Feeder (the Interpreter) is interpreting the Key Signature store in header which govern the way that the music should be played.