6.005 Design Milestone
Team Members: Srinidhi Viswanathan, Clare Liu, Yee Ling

Contacts:
Clare: clareliu@mit.edu 240-688-5319
Yee Ling: elainegn@mit.edu 857-253-9063
Srinidhi: srinidhi@mit.edu 503-329-7248

ABCMusic ANTLR:

Lexical Tokens:

```
REST : 'z';
DIGIT: [0-9]+;

BASENOTE : 'C' | 'D' | 'E' | 'F' | 'G' | 'A' | 'B'

        | 'c' | 'd' | 'e' | 'f' | 'g' | 'a' | 'b';

TITLE : 'T:' [ ']* [a-zA-Z\.\'0-9(),?' '-]*;
COMPOSER : 'C:' [ ']* [a-zA-Z\.\'0-9(),?' '-]*;
VOICE : 'V:' [ ']* [a-zA-Z\.\'0-9()?,' '-]*;
LYRIC : 'w:' [ ']* ([a-zA-Z\.\'\!0-9(),?' '-] | LYRICALELEMENT)* ;
COMMENT: '%' [ ']* [a-zA-Z\.\'\!0-9(),?' '-]*;
METER: 'M:' [ ']* NOTELSTRICT;
COMMONTIME : 'M:' [ ']* 'C';
CUTTIME : 'M:' [ ']* 'C|';
KEY : 'K:' [ ']* [A-G] KEYACCIDENTAL? MINOR?;

NOTELSTRICT : ([0-9]+'/'[0-9]+);

KEYACCIDENTAL  : '#' | 'b';
MINOR : 'm';
LINEFEED: '\n';
WHITESPACE : [ \t\r]+ -> skip ;
ACCIDENTAL : '=' |'^' | '^^' | '_' | '__';

TEMPO : NOTELSTRICT '=' DIGIT;
OCTAVE : ['']+ | [',']+;

NOTELENGTH : [0-9]*'/'[0-9]+;
NOTELENGTH2 : [0-9]*'/'+;
```

LEFTP   : '(';
BAR   : '|';
THINTHIN : '||';
THINTHICK : '|]';
THICKTHIN : '[|';
REPEATL : '|:';
REPEATR : ':|';
NTHREPEAT : '[1' | '[2';
LYRICALELEMENT : ' '+ | '-' | '_' | '*' | '~' | '\-' | '|';

Our Modified Grammar:
abc_tune : abc_header abc_music EOF;
abc_header : field_number comment* field_title other_fields* field_key;
field_number : 'X:' fieldnum* end_of_line;
//field_title : 'T:' title* end_of_line;
field_title : TITLE end_of_line;
other_fields : field_composer | field_default_length | field_meter | field_tempo | field_voice | comment;
//field_composer : 'C:' composer* end_of_line;
field_composer: COMPOSER end_of_line;
field_default_length : 'L:' notelengthstrict end_of_line;
field_meter : (METER | COMMONTIME | CUTTIME) end_of_line;
field_tempo : 'Q:' TEMPO end_of_line;
field_voice : VOICE end_of_line;
field_key : KEY end_of_line;
fieldnum: DIGIT;
temp: DIGIT;
temponote: notelengthstrict;
notelengthstrict: NOTELSTRICT;

abc_music : abc_line+;
abc_line : element+ LINEFEED (lyric LINEFEED)? | mid_tune_field | comment;
play_types : note | rest | chord | tuplet_element;
element : play_types | barline | NTHREPEAT | WHITESPACE;

note : pitch (notelength)?;
pitch : (ACCIDENTAL)? BASENOTE (OCTAVE)?;
rest : REST (notelength)?;

notelength: NOTELENGTH2 | NOTELENGTH | DIGIT | NOTELSTRICT;

tuplet_element : tuplet_spec note+;

tuplet_spec : LEFTP DIGIT;

chord : '[' note+ ']' (notelength)?;

endbar : THINTHIN | THINTHICK | THICKTHIN;

repeat : REPEATL note+ REPEATR;
barline : BAR | endbar | repeatbar;
repeatbar : REPEATL | REPEATR;

mid_tune_field : field_voice;
comment : COMMENT LINEFEED;
end_of_line : comment | LINEFEED;

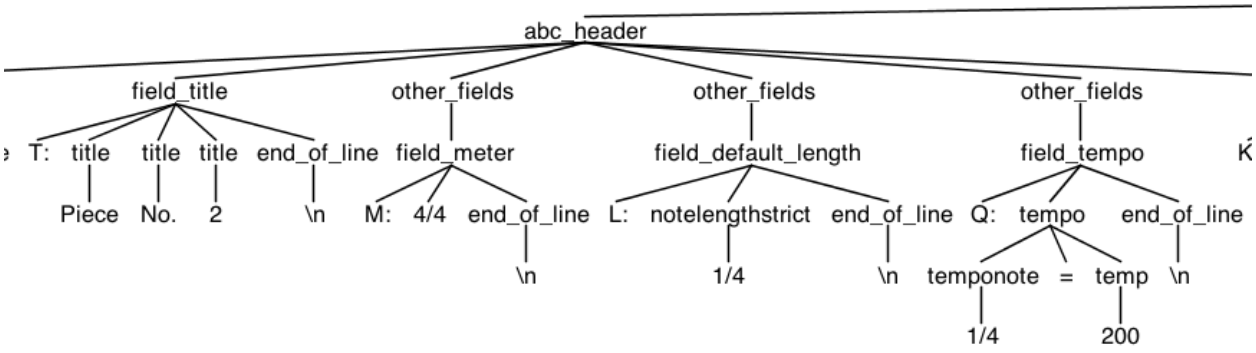
lyric : LYRIC;

Strategy for using ANTLR:
To create our grammar, we mainly followed the subset of ABC 2.1 in BNF format, taking care to format it properly in ANTLR. Some of the changes were:

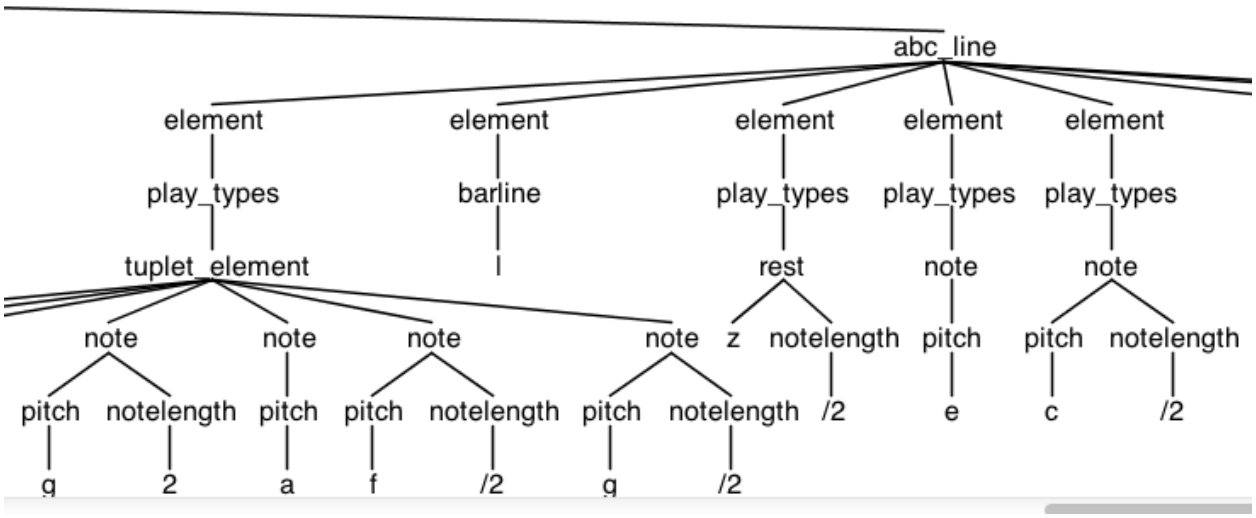
- Adding the nonterminal **play_types** to differentiate between 4 main types: note, rest, chord, and tuplet.
- Created tokens for several header fields, like composer, meter, title, etc.
- Changing the given grammar for “barline” to differentiate between the repeat bar “|:” and normal bar line “|”.
- **Handling errors:** ANTLR will automatically generate an error if it sees an unaccounted character (such as “@”) - unless it’s found in the title or composer, which will allow almost anything.
    - Also, if the input abc file is missing the header or body, we will throw an exception
    - Specifically, we will throw exceptions if the first field in the file is not X (index number), or if the second field is not T (title), or if the last field in the header is not K (key)

Parts of our parser tree from parsing piece2.abc is displayed below:

Tree under abc_header, which includes some of the nonterminals such as field_title and other_fields such as tempo and meter.



Tree under abc_line, which includes the primary notes which are categorized under play-notes (such as rest, note, and tuplet). Barline is also classified as an element, which will help to demarcate measures for the parser.



Our parser can successfully parse scale, piece1, piece2, piece3(minus lyrics).

Design and Representation of Input -> SequencePlayer:
Our lexer will not handle the structure of the language at all. It will only identify the text. It will be the job of the parser to separate the String value into any necessary parts. When going through the parser tree, we will keep track of everything between two barlines (seen under abc_line-element), which we will add to Measure (see datatypes). Each Measure will be added to a Repeat as it is completed. Finally a Repeat will be added to the corresponding Voice, along with the expected number of times it should be played. Finally, each Voices will be added to Music, which is the body of the Score.

Additional notes:
We will have to calculate LCM of the denominators to decide count/ticks

We will have a counter to keep track of the timestep the music is currently on.
Get the pitch from each note to play: transpose a note if there is an accidental or transpose octave up/down

Abstract Data Type Hierarchy

The entire abc file is made up of 2 parts: header and body. We will use an interface for Header and Body to organize our datatypes and Listener tree.

Score = Header + Body

Interfaces:

- **Header**
- **Body**
- **PlayType**
    - Methods:
        - add() - adds an event to be processed by SequencePlayer

Header classes:

- class Composer implements Header
    - Constructor:
        - public Composer(String composerName)
    - Immutable
    - Methods
        - String getComposer(): returns the name of the composer
- class Key implements Header
    - Constructor:
        - public Key(String key, String minor)
        - public Key(String key)
    - Immutable
    - Methods:
        - String getKey(): returns the key of the piece. ^ and _ will be used to represent sharps and flats and m will be used to represent minor
        - String[] getAccidentals: returns a list of notes that have sharps or flats in the key. For example, if the key is D major, this method will return an array that contains F# and C#.
- class Length implements Header
    - Constructor:
        - public Length(Fraction defaultLength)
    - Immutable
    - Methods:
        - Fraction getLength(): returns the default length
- class Tempo implements Header
    - Constructor:
        - public Tempo(Fraction temponote, int tempo)

    - Immutable
    - Methods:
        - Fraction getTempoNoteLength(): returns the default length of a note in the tempo
        - int getTempo(): returns the tempo of the piece
- class Meter implements Header
    - Constructor:
        - public Meter(Fraction meter)
        - // The listener will convert C to 4/4 and C| to 2/2
    - Immutable
    - Methods:
        - Fraction getMeter(): returns the meter of the piece
- class Title implements Header
    - Immutable
    - Methods:
        - String getTitle(): returns the title of the piece
- The track number (X:) will be discarded after reading, so we will not create a class for this.

We believe all the header fields are immutable since they should not change while the sequence is being played.

Body classes:

- class Music implements Body
    - Constructor:
        - public Music(List<Voice>)
    - Mutable
    - Methods:
        - Voice getVoice(String voiceName): returns the Voice that matches voiceName
- class Voice implements Body
    - Constructor:
        - public Voice(List<Pair<Repeat r, int numRepeats>>)
    - Mutable
    - Methods:
        - PlayType getRepeat(): returns a PlayType that represents the part of the music that is repeated
        - int getNoOfRepeats(int index): return an int representing number of times that a Repeat should be played
- class Repeat implements Body
    - Constructor
        - public Repeat(List<Measure>)
    - Mutable

    - Methods:
        - Measure getMeasure(int index): return Measure in Repeat that corresponds to index number
- class Measure implements Body
    - Constructor:
        - public Measure(List<PlayType>)
    - Immutable
    - Methods:
        - List<PlayType> getPlayTypes(): returns a list containing the play types in the measure
- class Note implements PlayType, Body
    - Constructor
        - public Note(BaseNote noteName, int octavesUp, Fraction noteLength, Accidental accidental)
    - Immutable
    - Methods:
        - get accidentalValue(): return the accidental value to transpose the note by
        - char getPitch(): return the pitch of a note
        - int getLength(): return the length of a note
        - int getTimestep(): return the timestep of a note
        - void add(BaseNote b, int timestep, int notelength): prepares a note to be added to SequencePlayer
- class Chord implements PlayType, Body
    - Immutable
    - Constructor:
        - public Chord(Note[] notes, Fraction notelength))
    - Methods:
        - Note[] getNotes(): returns an array of all notes that make up the chord
        - Fraction getTime(): returns the notelength of the entire chord
        - void add(Note[] notes): calls the add method of each note and keeps the timestep constant so that all notes in a chord are played at the same time
- class Rest implements PlayType, Body
    - Immutable
    - Constructor:
        - public Rest(String length, Int timestep)
    - Methods:
        - int getLength(): return the length of the rest
        - int getTimestep(): return the timestep of the rest
        - void add(): increments the timestep without adding a note
- class Tuplet implements PlayType, Body
    - Constructor:
        - public Tuplet(Note[] notes,int numNotes, int length)
    - Immutable

    - Methods:
        - Note[] getNotes(): returns an array of all notes that make up the tuple
        - void modifyNoteLength(): change the lengths of all of the notes in the tuple to add up to the length (mutator)
        - void add(Note[] notes): calls the add method of each note and increments the timestep by the modified note length for each note
- class Lyric implements Body, PlayType
    - Constructor:
        - public Lyric(String lyrics)
    - Methods:
        - add(String lyric, int timestep): prepares a lyric event to be added to SequencePlayer

Additional Helper Classes to help with storing data

- **class Accidental**
    - enum class to make it easy to read and identify accidentals
- **class BaseNote**
    - enum class to identify notes (A,B,C,D,E,F,G)
- **class Counter**
    - Class for counting timesteps, so that when note is added, the counter can be used as the startTime for the note
- **class Fraction** (int numerator, int denominator)**:**
    - Represents a fraction to keep track of notelengths and the meter
    - int getNumerator(): returns numerator of Fraction.
    - int getDenom(): returns denominator of Fraction
- **class Lcm**(List<int> denominators)
    - getLCM(): returns the lowest common multiple of all the denominators to set the number of ticks per beat
- **class Pair** (X first, Y second) : represents a pair so we can deal with measures and repeats

We think that music, voice and repeats should be mutable so that we can add on Voice, Repeat and Measure to the respective lists as we parse the music.

Common methods:
We will override toString(), equals(), hashCode() methods for each class that we implement. At this point, these are the methods we see as necessary.

Testing Strategy:

We will test our lexer, our parser and our play method to test the conversion of a parse tree into our datatypes. In addition, we will test equals(), toString(), and hashCode() methods.

Lexer: Test all tokens relating to the header, all tokens relating to the body, test strings that include the transition from the header to the body

Parser: Test tunes with lyrics and without lyrics, test tunes including repeats and not including repeats, test tunes with only 1 voice and with multiple voices, test invalid inputs such as invalid characters. Test when M, Q, L, C are omitted / not defined in the input.

Datatypes: test that all methods of each datatype work correctly, including toString(), equals(), and hashCode()