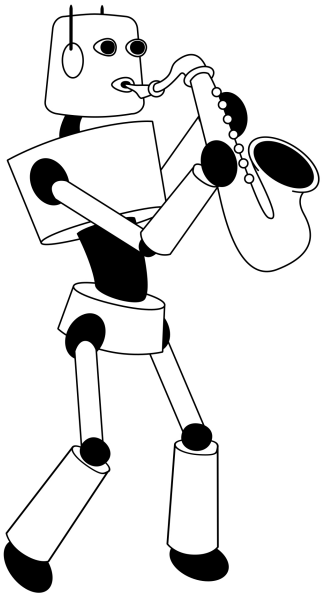
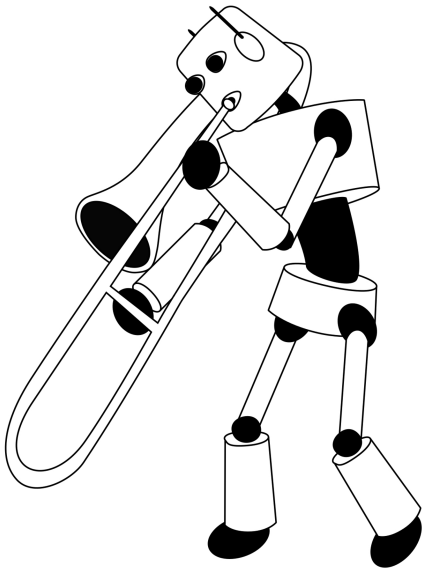
Contents

Addendum

https://bac.kgpl.org

The following chapters are only available as a pdf or online. They are not in the printed book.





Robots and cover art drawn by Kier Heyl.

About This Book

Although on the surface it appears that music and math have little in common there are many underlying similarities. Both are rigorous systems with an extensive history. Both blossomed with amazing variety in the 20th century. Both appeal to an organized mind seeking regularity in chaos.

There are also many differences. The mathematical proof is a core concept in mathematics that has no real counterpart in music. Well written proofs are said to be beautiful but not with the same meaning that we use when we say a piece of music is beautiful. And just like there are many people proficient in music and math there are also many musicians that have not had extensive math instruction and computer programmers who can't play a note.

Computer programming is closer to mathematics than it is to music. Most Computer Science Departments at universities grew out of the Math Department. Computer Science degrees require extensive math coursework and most often might require one, university wide, music course.

This book covers all three subjects, music, math, and computer programming. It is written to provide an introduction to algorithmic composition for composers interested in the field. I also provide music theory to programmers interested in applying their skills to music. Hopefully it provides enough music theory, computer science, and math to inform someone at the lay level in all three fields however they are not the intended audience.

If you are a composer interested in learning algorithmic composition this book is for you. If you are a mathematician who would like to realize your mathematical constructs as music this book is for you. If you are a computer programmer who would like a starting place for programming your own music this book is for you.

Whatever your field you will find that some parts of this book are exceedingly simple and others are like reading a foreign language. Your background will determine which parts are easy and which parts are hard. So skim past the easy parts and seek help on the hard parts from someone more versed in that area of study, if necessary. And please, have a good time doing it.

Introduction

This is a "work along with me" book. I will present algorithmic composition strategies for Bass, Melody, and Rhythm by working through example programs. Each section will use a different type of algorithm and address different areas of music theory.

In the Bass section the algorithm used is rule based and deterministic. The music theory covers the importance of the V I interval for bass players and jazz walking bass lines.

In the Melody section we look at using a database of note sequences. The algorithm involves random selection of melody notes restricted to the sequences stored in the database to output an endless stream of eighth notes. Then we make it better by adding rests and notes of longer duration.

In the Rhythm section we explore polyrhythms using prime number theory to generate unique rhythms of any length. Surprisingly enough this uses the simplest algorithm of all three sections. And provides the most variety.

In each section we implement the algorithm in code and then show how we can change the code or the data to tweak the composition. It is always easier to improve working code then to write a program from scratch.

Besides implementing the algorithms we also write programs to output the notes to musicXML which can be imported into most music composition programs for printing sheet music or listening to the compositions.

For further study there's an internet addendum also available as a pdf covering techniques being used in algorithmic composition including artificial intelligence.

The addendum also includes a Listener's Guide which might be the place you want to start. The Listener's Guide looks at some important pieces historically and also points to the future, where the real action in algorithmic composition will be happening.

Here's the link to the website hosting this book and the addendum.

https://bac.kgpl.org

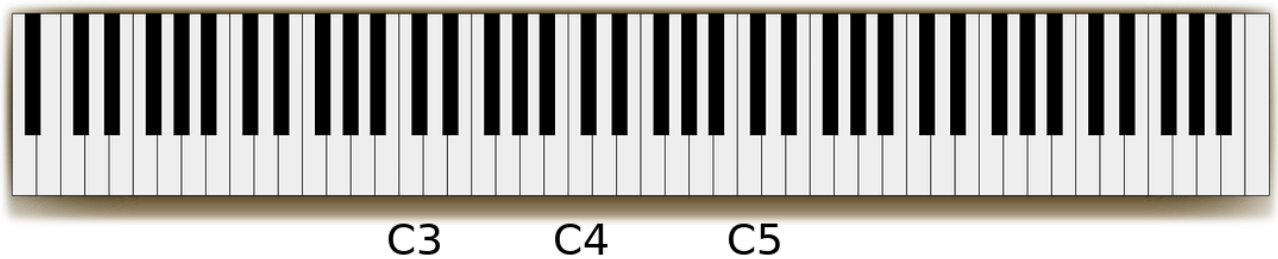
Music Theory And Notation

All composers study music theory and algorithmic composers more than most. If you don't have a background in music theory I recommend reading and/or course work. This book doesn't go very deep into music theory so I will cover the necessary fundamentals here and expand on them as we work on the algorithms.

If you are a musician feel free to skim or skip this section. If you have no familiarity with music theory you may find this text quite dense and I recommend that you sit down at a piano keyboard and read through it slowly playing and listening as you read.

The Keyboard

The piano keyboard is the best tool for understanding music theory. Looking at the piano we see white keys and black keys. The white keys play the notes A-B-C-D-E-F-G. The black notes are sharps or flats, each black note has two names. Looking at the black notes we see they are arranged in groups of two and three. The white note immediately to the left of a group of two black notes is C. The C note in the center of the keyboard is called Middle C or C4. The next C note to the right of Middle C is called C5. C5 is higher in pitch than C4 and it is said to be an octave above C4. The next C note to the left of Middle C is called C3. C3 is lower in pitch than C4 and it is said to be an octave below C4.



The black note immediately to the right of Middle C is called C# (read C sharp). The black note immediately to the left of D is called Db (read D flat). C# and Db are the same note! This is called enharmonic equivalence. All the black notes are named the same way. They are the sharp of the white key to their left and the flat of the white key to their right.

Scales

A scale is a row of notes. The C major scale consists of the notes C-D-E-F-G-A-B-C. Starting at Middle C play all the white notes going up in pitch until you reach the next C note and that is a C scale.

Scales can also descend. Play all the white notes from Middle C down to the next C note and that is the descending C scale, C-B-A-G-F-E-D-C.

Intervals

Look at the C major scale, C-D-E-F-G-A-B-C. We number these notes 1, 2, 3, 4, 5, 6, 7. C is 1, D is 2, etc. Since this is a C major scale the note, C, is called the root. The interval between C and D is called a second. The interval between C and E is called a third. The interval between C and F is called a fourth. And so on. The first note in the interval is C or the root. The names of the intervals correspond to the position of second note in the interval.

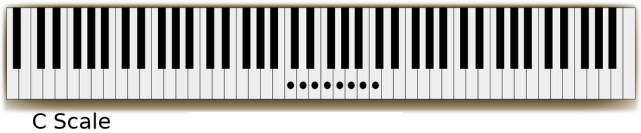
When two notes are adjacent on the keyboard we call that interval a half step. Since E and F are adjacent the interval between E and F is a half step. Since C and C# are adjacent that interval is also called a half step.

When two notes are two half steps apart we call that a whole step. C to C# is a half step and C# to D is a half step so C to D is a whole step. With the root as the first note the interval of a whole step is also called a second. Similarly the interval of a half step is called a diminished second. An interval of three half steps, C to Eb for example, is called a minor third.

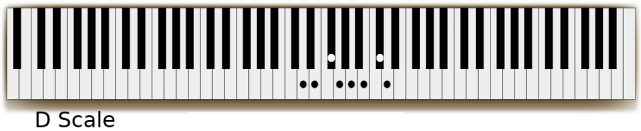
Music theory wasn't planned and the way musical ideas are named just grew willy-nilly in different countries and different cultures at different times. Often the same thing can be said in many ways. Sometimes the meaning is exactly the same. Sometimes nearly the same. Sometimes, with these music terminology synonyms one or the other is considered more correct depending on the context. In an attempt to keep things understandable we will not go into all the names we can call these basic concepts.

Scales As Intervals

Above I said a scale is a row of notes. But a scale can also be defined as a set of intervals. Looking at the C Major scale C-D-E-F-G-A-B-C and listing all the intervals we get whole step, whole step, half step, whole step, whole step, whole step, half step, seven intervals between the eight notes. This can be thought of as the definition of a major scale. If we want to play a D major scale we start on D and then following the intervals listed above we get D-E-F#-G-A-B-C#-D. The sharped notes, F# and C#, are the necessary adjustment made to get the whole note and half note intervals needed for a major scale. Using this set of intervals we can construct all 12 major scales, seven starting on white keys and five starting on black keys.



C Scale



D Scale

Church Modes

A mode of a scale is a scale that starts on a note that is not the root. If we take a C major scale and start it on D instead of C we get D-E-F-G-A-B-C-D. Play a C major scale. All the white keys form C to C. Now play all the white keys between D and D. Hear the difference? When playing from D to D we hear a minor scale. Same notes. Different sound.

The Church Modes are all the variations of a major scale using each of the seven pitches as a starting point. It is easy to play these in the key of C because all seven church modes use the white keys on the piano. Play the white notes from E to E. Now that that might sound different or foreign to some. Scales that start with a half step are not common in American popular music and it takes some time to become comfortable with the sound. The church modes that start on F, G, and A are common in all types of music and should sound familiar to you. The final church mode in the key of C starts on a B and again, it will probably sound different at first.

Each church mode has a name. For example the church mode in the key of C that starts on an A is called both Aeolian or Natural Minor. There are also many other scales that are not church modes like Harmonic Minor, Chromatic, and the Bebop scale.

Chords

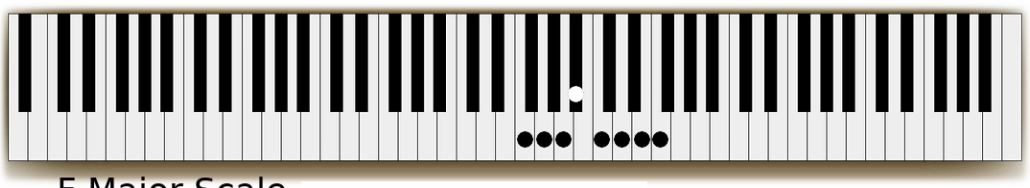
Just like two notes define an interval three or more notes define a chord. Just like scales can be thought of as a list of intervals chords can be thought of as a list of intervals. In the key of C major skipping every other note results in C-E-G which is called a C major triad. The intervals in a major triad starting with the root note are 4 half steps also called a major third and then 3 half steps, called a minor third. Triads can also be built on minor chords, for example D-F-A is a minor triad built on a D minor scale.

If you add one more note to a triad, skipping a note like before, we get a seventh chord. Here is a list of the seventh chords built on the church modes in the key of C and the most common name for these chords and a frequently used chord symbol or abbreviation for these chords.

```
C-E-G-B - C major seventh ----------- Cmaj7
D-F-A-C - D minor seventh ----------- Dmin7
E-G-B-D - E minor seventh ----------- Emin7
F-A-C-E - F major seventh ----------- Fmaj7
G-B-D-F - G dominant seventh -------- G7
A-C-E-G - A minor seventh ----------- Amin7
B-D-F-A - B half diminished seventh - Bmin7b5
```

These are the chords used in the example program in the Bass section of this book. Except in the key of F.

The F major scale is F-G-A-Bb-C-D-E-F



F Major Scale

And the seventh chords in the church modes of the F major scales are.

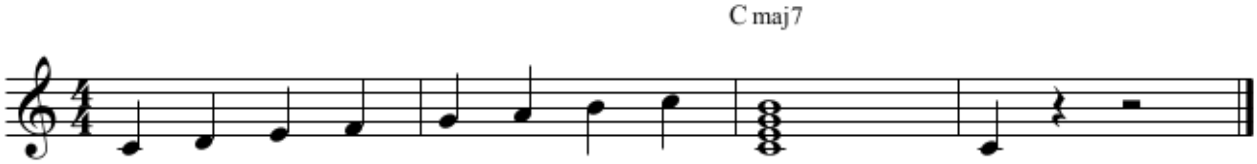
```
F-A-C-E -- F major seventh ----------- Fmaj7
G-Bb-D-F - G minor seventh ----------- Gmin7
A-C-E-G -- A minor seventh ----------- Amin7
Bb-D-F-A - Bb major seventh ---------- Bbmaj7
C-E-G-Bb - C seventh ----------------- C7
D-F-A-C -- D minor seventh ----------- Dmin7
E-G-Bb-D - E half diminished seventh - Emin7b5
```

It is easy to see the similarities between these two tables. Since they are both built on major scales the intervals are the same and the chord qualities (major, minor, diminished) are the same too.

Music Notation

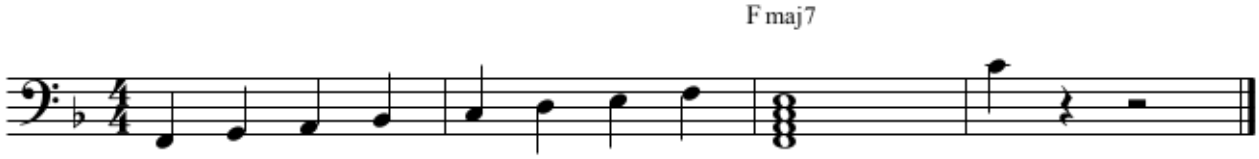
Most music is written on a staff of five lines, treble clef for higher pitched instruments and bass clef for lower pitched instruments.

Notes are placed on the lines and between the lines. The bottom line of the treble clef is an E4 so Middle C or C4 is on the first line below the staff.



The 4/4 notation at the beginning means 4/4 time or 4 beats per measure with a quarter note for each beat. The scale notes are quarter notes and the chord notes are whole notes, a note with the same duration as 4 quarter notes. The last bar has a Middle C quarter note, a quarter rest, and a half rest.

The top line on the bass clef is A3 so Middle C is one line above the staff. Counting down from the A3 we get G2 on the bottom line and that puts F2 in the first space below the staff.



The flat near the beginning is called the key signature and it means that we are in the key of F which has one flat. It is placed on the B note to designate Bb. Musicians will now read all of the B notes as Bb because they know that the music is written in the key of F. Again the last bar starts with a Middle C quarter note.

This is not a book about music theory. I have only covered the theory necessary to understand the algorithms used in the examples.

For a readable explanation about music notation I recommend "Henscratches and Flyspecks" by Pete Seeger. He has a way of explaining this rather technical content in an entertaining way.

Notes on a staff - Pete Seeger "Henscratches and Flyspecks"
https://www.goodreads.com/book/show/686464.Henscratches_and_flyspecks

Names for notes in different octaves
http://openmusictheory.com/pitches.html

musictheory.net - free online content
https://www.musictheory.net/

Understanding Music Theory
https://www.earmaster.com/music-theory-online/course-introduction.html

Computer Programming

It is not necessary to do computer programming to do algorithmic composition. If your algorithms are simple, as they should be with your first algorithmic compositions, it is possible to do the work by hand without ever using a computer. Historically speaking algorithmic composition predates computers by many centuries and often the algorithms employed weren't even thought of as algorithms but were just the way the songs were sung.

That said, modern algorithmic composers almost always use computers generating their compositions. The process of writing psuedo code defining an algorithm and then coding it in a computer language is a great way to really understand what an algorithm is and what algorithms can do. If you have not done any programming I recommend that you take a class and read on the internet. Beginning programming is taught at almost every university and high school and there are many websites devoted to learning programming with specific examples. I use some of these websites, including w3schools.com, with every programming project I do.

This is not a book about computer programming. I am going to cover some elementary computer concepts in this chapter that will be used in the example programs. If you are a programmer feel free to skim or skip this chapter. For readers without programming experience I will keep this chapter as simple as possible to expose you to some of the techniques used in thinking about and coding the example algorithms. Since the example programs are written in the php programming language I will use php in this chapter as well.

Plain Text

Programs are written in text files called plain text to distinguish them from word processor files that also include formatting instructions to display different fonts, sizes, headers, footers, etc.

Text editors or programming editors are used to edit and save plain text files, most commonly with the extension .txt. Files containing php code usually have the extension .php but they are still plain text files. XML and muxicXML data files are also plain text often with the extensions .xml or .musicxml.

Word processor files often have extensions like .rtf, .odt, or .doc. Word processor files will not work for any of the example programs in this book. Most word processors, however, do support plain text output and you can use a word processor as long as you are careful to specify text or plain text when you save.

Variables And Assignment

Variables are used to store values. In the php programming language variables start with a dollar sign. So you might see things like this.

```
$counter = 0;
```

The variable $counter is set to equal the number 0.

or

```
$chord = "Cmaj7";
```

The variable $chord is set to equal the characters Cmaj7. The quotes around Cmaj7 let the program know that you are storing characters.

To display the contents of a variable in php you use the echo statement.

```
echo $counter; // displays 0

echo $chord; // displays Cmaj7
```

The two forward slashes tell php that what follows is a comment and should not be executed.

Conditionals

Conditionals implement if then logic like this.

```
if(condition){
  do something here
}
```

or

```
if(condition){
  do something here
}else{
  do something else here
}
```

Indentation is commonly used writing code to make it easier to read conditionals and loops.

Here's a php example that does something. It uses the php rand() function which is used in the example algorithm in the Melody section. In this example we flip a coin. Notice the two equal signs used to compare values. One equal sign is for variable assignment like above.

```
<?php

// Set a random number to 0 or 1
$randomnumber = rand(0,1);

// flip a coin
if($randomnumber == 1){
  echo "heads";
```

```
}else{
  echo "tails";
}

?>
```

This program, coinflip.php, is in the programming folder in the supplemental materials.

There are more conditionals like if then elseif constructs and switch case constructs for more complicated tasks.

Loops

The most common loop is called a while loop. It is a loop with a built in conditional.

```
while(conditional)
  do stuff
end while
```

Loops terminate when the conditional is false. A common error writing loops is to write a conditional that never becomes false. Loops that don't terminate are called infinite loops and you will wait a long long time for your infinite loop to complete.

Here's a php example.

```
<?php

// assign initial values
$topvalue = 100;
$counter = 0;

// conditional
while($counter < $topvalue){

  // increment (add 1 to) $counter
  $counter++;

  // display the counter value
  echo $counter;
}

?>
```

This is an actual program that you can run in php that will count the numbers from 1 to 100. If I hadn't coded the increment counter line I would have created an infinite loop. The program, loop.php, is in the programming folder in the supplemental materials.

Reading And Writing Text Files

An important part of the example programs is reading data from a text file, running an algorithm on this input data, and writing the results to an output text file. The output file will sometimes be musicXML which can be imported into sheet music programs.

Pseudo Code

Pseudo code is important to developing programs that implement algorithms. As an example I have written some pseudo code to explain reading and writing text files. Every program in the book does this in php so you can see the actual php code to read and write files in the example programs.

```
Open an input text file for reading.
Open an output text file for writing.
Read a line from the input file into a variable.
Write the variable to the output text file.
Close the input text file.
Close the output text file.
```

Notice that you have to explicitly open and close the files.

What this pseudo code describes is a program that reads the first line from an input file and writes only that line to an output file. If there is more than one line in the input file and you want all of them in the output file use a loop. If you only want some of the lines in the input file written to the output file use a conditional in the loop.

Arrays

An array is a table of values. It's kept in memory and only exists while the program is running, unlike text files that will remain on the computer after the program is finished.

Here's the simplest example in pseudo code. We create an array, colors, like this

```
1 -> Blue
2 -> Red
3 -> Green
```

Then

```
echo colors[2];
```

displays Red

```
echo colors[3];
```

displays Green

You can loop through arrays, sort arrays, and build arrays with more complicated structures. In the example programs in this book we only use this very simple type of array.

php is free open source software that runs on windows, mac, and linux.
https://www.php.net/

W3schools teaches php and other programming languages online.
https://www.w3schools.com/

MusicXML And Finale

MusicXML is a standard data format for describing sheet music. Most sheet music editing programs import and export musicXML format making it possible to move files form one program to another with minimum fuss and data loss. Like most things having to do with computers musicXML is not perfect but it is steadily improving. The code in this book will implement the algorithms and output musicXML. So when working with this book it's fine to use any sheet music editor that supports musicXML files. Just like musicXML is not perfect the code that imports and exports musicXML sometimes has flaws too. This is a truism for any complex data application. Humans are not perfect and neither are the programs they write.

MusicXML has a website.
https://www.musicxml.com

On this website there is a FAQ page.
https://www.musicxml.com/tutorial/faq

On the FAQ page there is an explanation of the musicXML file extensions.

MusicXML 3.1 has two registered media types:

application/vnd.recordare.musicxml for compressed .mxl files
application/vnd.recordare.musicxml+xml for uncompressed .musicxml files

Note that the FAQ specifies version 3.1. Prior to that musicXML used the .xml extension. Either extension is fine. I recommend using .xml for version 3.0 and before and using .musicxml for version 3.1 and future versions. Still, either works. Just like a JPEG standard graphics file can use either extension .jpg or .jpeg.

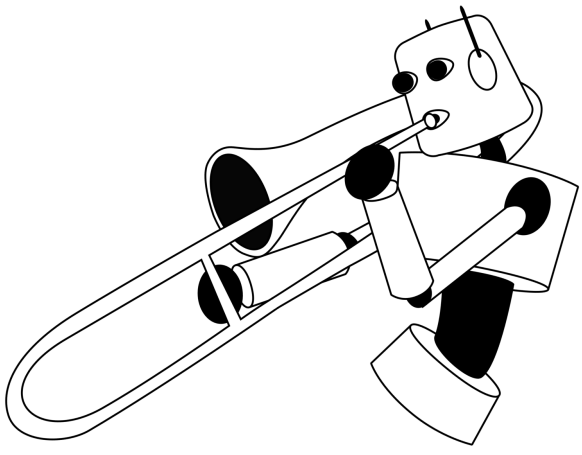
I am using Finale v26 which supports musicXML 3.1 out of the box. If you have Finale 2012 or earlier you can install the Dolet 7 musicXML plugin for Finale which provides version 3.1. If you can use the current version I recommend it. But as far as the exercises in this book are concerned it doesn't matter. Everything will work with either version 3.0 or 3.1.

Finale is a sheet music program published by makemusic. There are many other sheet music programs and like I said above any of them will work fine for the exercises in this book as long as the program supports musicXML.

So when I say Finale in this book I mean Finale or your sheet music program. I just use Finale as a shorthand since that's the program I used writing the book.

If you don't have a sheet music program you can download Finale Notepad for free. It has musicXML support and will work for the exercises in this book.

The robots and the cover art were drawn by Kier Heyl.



Bass

The most important interval in music is the V-I or fifth interval. This interval is said to establish the I chord as the tonic chord. In country music or folk music this is often seen in the change I-IV-I-V-I. In the blues we often get I-I-IV-I-V-I, essentially the same change except for the timing. The most common change in jazz is ii-V-I, again we see the V-I establishing the I as the tonic of the key. The lower case ii means the chord is minor. The importance of the V-I interval is true not just for 20th century music. Throughout music history the V-I interval has played the same definitive role.

For those reading this who don't know what V-I means we will look at the key of C. The major scale in the key of C is C-D-E-F-G-A-B-C. C is the first note and the root of the I chord. G is the fifth note and the root of the V chord. So G to C is called the V-I chord progression.

A measure and a bar are the same thing. In 4/4 time there are 4 quarter notes in a bar and two bars are counted 1-2-3-4 1-2-3-4. In 3/4 time there are 3 quarter notes in a bar and two bars are counted 1-2-3 1-2-3. In 6/8 time there are 6 eighth notes in a bar and two bars are counted 1-2-3-4-5-6 1-2-3-4-5-6. In this book the example algorithms are all written in 4/4 time.

Bass players almost always play the root on the first beat of the measure. If they are counting 1-2-3-4 the first beat will be the root. In folk and country music this is often held for 2 beats or a half note. Then the other half note in the bar will be played on the fifth note of the scale. There are many top selling country songs where the bass player follows the chord change playing 1-5-1-5 all the way through the song.

Swing jazz bass players often play what is called a walking bass. A walking bass part has quarter notes played on every beat. Most jazz bass players are very creative with their walking bass lines but they still, almost always, play the root on beat 1. Beginning jazz bass players are often taught a formula bass line like 1-2-3-5 for measures with only one chord or 1-5-1-5 for measures with two chords.

After an aspiring jazz bass player learns to follow the chord change this way they will be taught how to vary this bass line and when it might be a good thing to do so.

For our algorithmic bass line example all of the chords will cover an entire bar and the bass part will be either 1-2-3-5, that is the first note of the scale defined by the chord, then the second note, the third note, and the fifth note. Or the bass will play 1-7-6-5, the root, down to the 7th, 6th, and 5th notes of the scale. So both the bass line and the chord change are simplified compared to many jazz standards in order to make the example algorithm work.

Rules For Deriving A Jazz Bass Part From The Chord Change

Simplification

Key of F.
Chords are seventh chords in the church modes. These are all the seventh chords containing only notes from the F major scale.
Chords change once a bar.
32 bar change.
Bass range E2 to A3, notes on the bass staff down to the E2 below the staff.
Quarter note bass parts.

The input file will be plain text with 32 lines and chord notation.

```
Amin7
Dmin7
Emin7b5
...
Fmaj7
```

Initial output will be a text file with one note per line. Flats and sharps are written b and #. The numbers are octave designators. C4 is middle C.

```
A2
Bb2
C3
E3
D3
...
```

Rules

Always play the root on one.

If the root is C or higher walk down 1-7-6-5.
If the root is B or lower walk up 1-2-3-5.

If there is a choice of notes for the root
  If this is the first note play the lower root
  If your last note played is C or higher play the higher root
  If your last note played is B or lower play the lower root

All Possible Chords And Start Notes With Quarter Note Parts

Because of the restriction to use only the church modes all of the chords consist of notes in the F major scale and there is only one chord available for each root note. Starting from the lowest note in our defined bass range the start notes, chords, and bass lines are as follows.

```
E1 -- Emin7b5 - E1,  F1,  G1,  Bb1
F1 -- Fmaj7 --- F1,  G1,  A1,  C2
G1 -- Gmin7 --- G1,  A1,  Bb1, D2
A1 -- Amin7 --- A1,  Bb1, C2,  E2
Bb1 - BbM7 ---- Bb1, C2,  D2,  F2
C2 -- C7 ------ C2,  Bb1, A1,  G1
D2 -- Dmin7 --- D2,  E2,  F2,  A2
E2 -- Emin7b5 - E2,  D2,  C2,  Bb1
F2 -- Fmaj7 --- F2,  E2,  D2,  C2
G2 -- Gmin7 --- G2,  F2,  E2,  D2
A2 -- Amin7 --- A2,  G2,  F2,  E2
```

Decision Tree For The First Note Of Bar

If this is the first note played play the lower root.
If your last note played is C2 or higher play the higher root.
If your last note played is B1 or lower play the lower root.

We know the root of the next chord and the last note played.

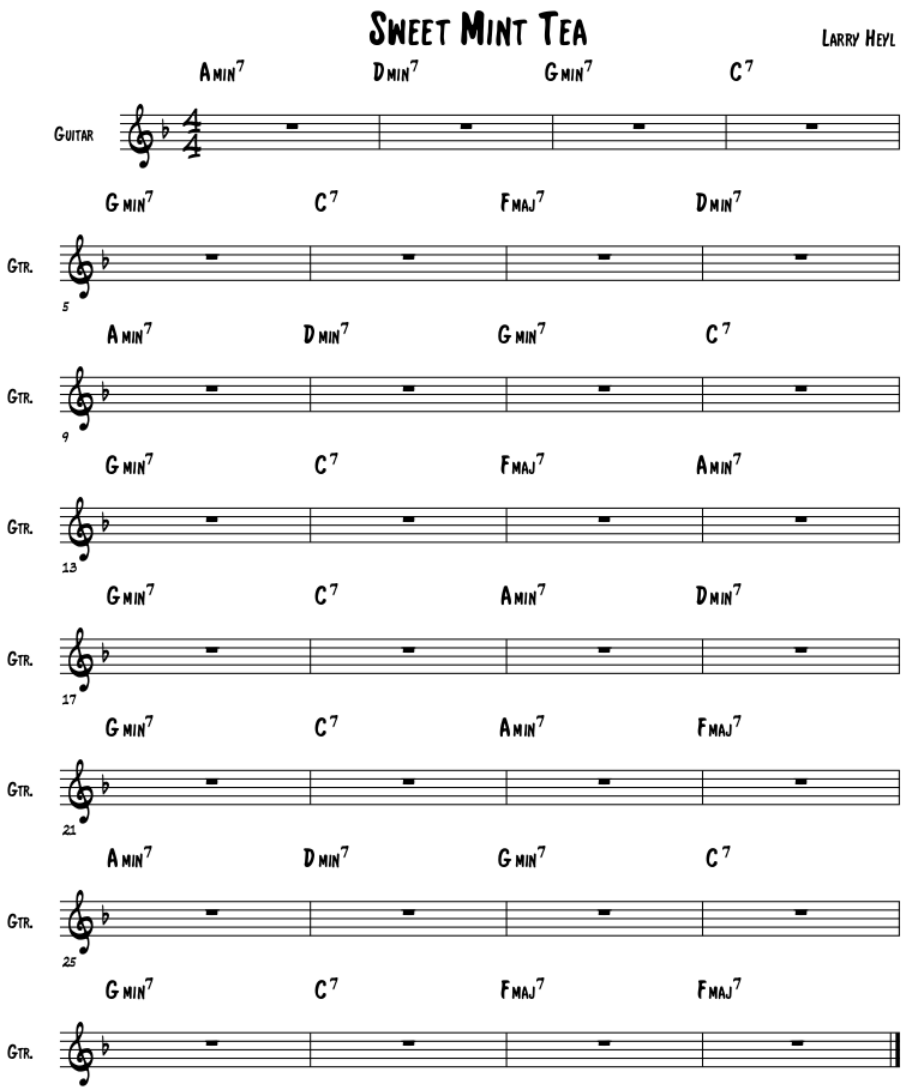
```
If root is Bb, C, or D
  Bb -> Bb1
  C -> C2
  D -> D2

If root is E, F, G, or A
  If last <= B1
    E -> E1
    F -> F1
    G -> G1
    A -> A1
  If last >= C2
    E -> E2
    F -> F2
    G -> G2
    A -> A2
```

Here is the data set for all the 4 beat bass lines.

E1	F1	G1	Bb1
F1	G1	A1	C2
G1	A1	Bb1	D2
A1	Bb1	C2	E2
Bb1	C2	D2	F2
C2	Bb1	A1	G1
D2	C2	Bb1	A1
E2	D2	C2	Bb2
F2	E2	D2	C2
G2	F2	E2	D2
A2	G2	F2	E2

Bass Implementation

I wrote a song, "Sweet Mint Tea", that uses only church modes in the key of F.



Pseudo Code For Generating The Bass Line

Create the input plain text file writing one chord per line.

Set the number of bars.

Create a data structure that maps root notes to 4 bass notes.

Open the input file for reading.

Loop through the input file one line at a time.

Using the decision tree above find the bass notes that match the chord just input and write them to the output string.

End the loop.

Write the output string to a file.

Code For Generating The Bass Line

Working code, makebass.php, is included with the book's supplemental materials available at bac.kgpl.org. To run this program and generate a bass line copy sweet_mint_tea.txt and makebass.php into a folder. Open a terminal or command prompt and type this command.

```
php makebass.php
```

Then hit enter.

To use another input file or to change the output file change the lines at the top of makebass.php shown below. Here's the actual program code.

```
<?php

// Set the number of bars.
$numberofbars = 32;

// Initialize output string.
$output = "";

// Initialize last note played.
$last = "";

// Set input file.
$inputfile = "sweet_mint_tea.txt";

// Open input file for reading.
$infile = fopen($inputfile,"r");

// Set output file.
$outputfile = "bassline.txt";

// Open output file for writing.
$outfile = fopen($outputfile,"w");

// Initialize variables mapping root notes to bass lines.
```

```
// Notice each chord is followed by a space, even the fourth one.
$E1 = "E1 F1 G1 Bb1 ";
$F1 = "F1 G1 A1 C2 ";
$G1 = "G1 A1 Bb1 D2 ";
$A1 = "A1 Bb1 C2 E2 ";
$Bb1 = "Bb1 C2 D2 F2 ";
$C2 = "C2 Bb1 A1 G1 ";
$D2 = "D2 C2 Bb1 A1 ";
$E2 = "E2 D2 C2 Bb1 ";
$F2 = "F2 E2 D2 C2 ";
$G2 = "G2 F2 E2 D2 ";
$A2 = "A2 G2 F2 E2 ";

// Loop through input file one line at a time.
while(! feof($infile)){
  // If this is the first note set last note played to the lowest note so we start
in the bottom octave of the range.
  if($last == ""){
    $last = "E1";
  }else{
    // Read the last note played from the output string.
    $last = trim(substr($output,-3));
  }

  // The root is the first character in the chord or the first two characters if
there are sharps or flats.
  // Get the chord.
  $chord = fgets($infile);

  // Get the root.
  $root = substr($chord,0,1);

  // If there is a sharp or flat add it.
  if(substr($chord,1,1) == "#" || substr($chord,1,1) == "b"){
    $root = $root.substr($chord,1,1);
  }

  // Check for each possible root.
  // Bb, C, and D only have one possible root.
  if($root == "Bb"){
    $output = $output.$Bb1;
  }
  elseif($root == "C"){
    $output = $output.$C2;
  }
  elseif($root == "D"){
    $output = $output.$D2;
  }

  // The rest of the chords have two possible roots.
  // Select the one to use based on the octave of the last note played, $last.
  elseif($root == "E"){
    if(substr($last,-1) == "1"){
      $output = $output.$E1;
    }else{
      $output = $output.$E2;
    }
```

```
  }
  elseif($root == "F"){
    if(substr($last,-1) == "1"){
      $output = $output.$F1;
    }else{
      $output = $output.$F2;
    }
  }
  elseif($root == "G"){
    if(substr($last,-1) == "1"){
      $output = $output.$G1;
    }else{
      $output = $output.$G2;
    }
  }
  elseif($root == "A"){
    if(substr($last,-1) == "1"){
      $output = $output.$A1;
    }else{
      $output = $output.$A2;
    }
  }
}

// Replace spaces in $output with newline character "\n".
$output = str_replace(" ","\n",$output);

// Write output.
fwrite($outfile, $output);

// Close files.
fclose($infile);
fclose($outfile);

?>
```

After you run this program you will have a file, bassline.txt, with one note per line.

MusicXML For The Bass Part

MusicXML is a standard XML data format used to describe sheet music. Like all XML files musicXML uses plain text and can be read and modified in a text editor. Most of the time musicXML will be written by one program and then read and rendered by another program. The bass2xml.php program will output a musicXML file that can then be read by Finale or any sheet music program with a musicXML import feature.

XML is not a programming language. It looks like code but XML is a data format. Since XML is plain text it is easily read and written by computer programs. So when you import a musicXML file into Finale it is Finale's import program that does all the work. The XML file has the data so the import program knows what clef, time signature, key, notes, dynamics, articulation, etc. to put on the page.

Just like programming code the XML files can become complicated and it is best to start with a working example and then modify it slowly, one thing at a time, until you eventually have the output you want.

Fortunately musicXML has a "Hello World example" in their tutorial.
https://www.musicxml.com/tutorial/hello-world/

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD musicXML 3.1 Partwise//EN"
    "http://www.musicXML.org/dtds/partwise.dtd">
<score-partwise version="3.1">
  <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>
```

Copy the text above and paste it into a text file called onenote.musicxml. This file is in the bass folder in the supplemental materials available at bac.kgpl.org.

Open it with Finale or any music composition program that has musicXML import. Soundslice has an online musicXML viewer here.
https://www.soundslice.com/musicxml-viewer

You should see one bar of sheet music with a single whole note. We will make a series of changes to this file to turn it into a bass line.

So here is what we see when we import onenote.musicxml:

Change this line

```
          <fifths>0</fifths>
```

to read

```
          <fifths>-1</fifths>
```

The 0 means no sharps or flats, key of C. The -1 means one flat, key of F. If we had used 1 it would mean one sharp, key of G. I saved this file as onenote-a.musicxml

When we import this changed file we see:

Now change these lines

```
          <sign>G</sign>
          <line>2</line>
```

to read

```
          <sign>F</sign>
          <line>4</line>
```

This changes the clef to bass clef and when we import this we see:

This is the only change that is not intuitive. I had to google this to figure it out. I saved this file as onenote-b.musicxml.

Next we change the line that reads

```
          <octave>4</octave>
```

to read

```
          <octave>3</octave>
```

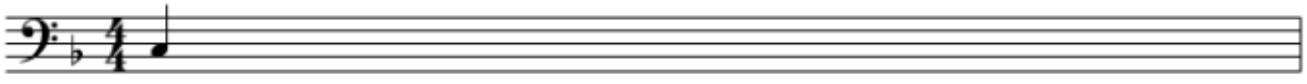
This changes the C from middle C to one octave below middle C and looks like this. I saved this version as onenote-c.musicxml.

Now find the lines that read

```
        <duration>4</duration>
        <type>whole</type>
```

change them to read

```
        <duration>1</duration>
        <type>quarter</type>
```



I saved this as onenote-d.musicxml.This changes the whole note to a quarter note and will produce this output. The change to duration was easy. I guessed right on

changing whole to quarter. Data formats are not always intuitive but in this simple example everything seems to be pretty logical. This is a plus for musicXML.

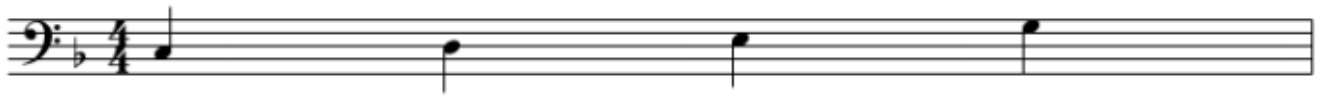
Find the note block that starts <note> and ends </note>

```
      <note>
        <pitch>
          <step>C</step>
          <octave>3</octave>
        </pitch>
        <duration>1</duration>
        <type>quarter</type>
      </note>
```

Copy this and paste it into the document 3 time immediately following the C note block shown above. Going down the note blocks leave the first one on C. Change the second one to D.

```
          <step>D</step>
```

Then change the third one to E and the fourth one to G.



I saved this as onenote-e.musicxml. When imported you will see this.
Now we see 4 quarter notes displaying a bass part starting on C. The filename, onenote.musicxml, is no longer correct so I am changing over to bass.musicxml. If you are following along you can rename the file or copy it to a new file called bass.musicxml.

There's one more step to complete this example and discover all the information we need to write a program that will output musicXML quarter note walking bass parts.

Find the measure block that starts <measure> and ends </measure>

```
    <measure number="1">
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>-1</fifths>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>F</sign>
          <line>4</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>C</step>
          <octave>3</octave>
```

```
        </pitch>
        <duration>1</duration>
        <type>quarter</type>
      </note>
      <note>
        <pitch>
          <step>D</step>
          <octave>3</octave>
        </pitch>
        <duration>1</duration>
        <type>quarter</type>
      </note>
      <note>
        <pitch>
          <step>E</step>
          <octave>3</octave>
        </pitch>
        <duration>1</duration>
        <type>quarter</type>
      </note>
      <note>
        <pitch>
          <step>G</step>
          <octave>3</octave>
        </pitch>
        <duration>1</duration>
        <type>quarter</type>
      </note>
    </measure>
```

Copy this text block and paste it immediately below resulting in two identical measure blocks.

Like before leave the C note and change the rest of the notes in the second measure to B, A, and G. The C note is still in the third octave but the other notes should be changed to the second octave by changing the 3 in the octave block to 2.

Since we are in the key of F we want to play a B flat. Musicians learn to do this automatically but musicXML doesn't. So we have to add an alter block like this under the B step block.

```
<alter>-1</alter>
```

This will change the B to a B flat.

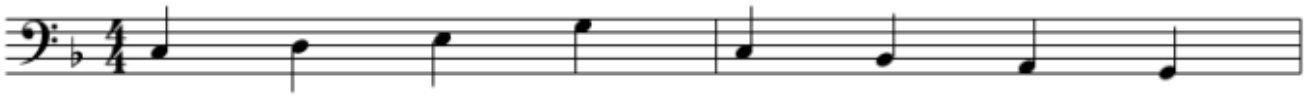
Also change the measure number from "1" to "2".

Here is the altered second measure.

```
    <measure number="2">
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>-1</fifths>
        </key>
        <time>
          <beats>4</beats>
```

```
        <beat-type>4</beat-type>
      </time>
      <clef>
        <sign>F</sign>
        <line>4</line>
      </clef>
    </attributes>
    <note>
      <pitch>
        <step>C</step>
        <octave>3</octave>
      </pitch>
      <duration>1</duration>
      <type>quarter</type>
    </note>
    <note>
      <pitch>
        <step>B</step>
        <alter>-1</alter>
        <octave>2</octave>
      </pitch>
      <duration>1</duration>
      <type>quarter</type>
    </note>
    <note>
      <pitch>
        <step>A</step>
        <octave>2</octave>
      </pitch>
      <duration>1</duration>
      <type>quarter</type>
    </note>
    <note>
      <pitch>
        <step>G</step>
        <octave>2</octave>
      </pitch>
      <duration>1</duration>
      <type>quarter</type>
    </note>
  </measure>
```

When the modified musicXML file is imported it looks like this.



Since our algorithmically generated bass part will be all walking quarter notes this example now shows all the musicXML text necessary to write the part. The next step will be to write a program, bass2xml.php, to output a file called bassline.musicxml that can be imported and then printed or played.

This example is designed to be easy. Almost everything is intuitive and easy to understand. MusicXML files, in general are not this simple. If you load a more complex score into Finale and then export it to musicXML you will see just how complicated musicXML can become.

Since even this simple example took quite a bit of work just to write 2 bars with 4 quarter notes in each it's easy to understand why most musicXML files are generated by computer. And that is exactly what we will be doing next.

Reverse Engineering A MusicXML File

Instead of starting with the Hello World template like we did in the example above we will do a little bit of reverse engineering.

I start by going to Finale and creating a 2 bar piece for piano bass clef using the Setup Wizard that has a walking bass line like the example above. I create a piece for piano and then delete the treble clef with the staff tool. After I save this piece of sheet music I then export it to musicXML using Finale's Export feature. I called these files bass_template.musx and bass_template.musicxml. Be sure to select the .musicxml extension, not the default .mxl extension. The .mxl files are compressed, not plain text, and they won't work for these programs.

This musicXML file has a lot more information than the Hello World file and will produce the sheet music that we really want.

Create a file called bass2xml.php.

Add the php opening and closing tags.

```
<?php

?>
```

In between the tags copy and paste the exported musicXML file. As a first step we will write the code that exports this file unchanged.

Like many things having to do with computers musicXML is persnickety about getting everything exactly right. As always we will write an easy program that works and then modify it a step at a time until we get the program that generates the output we really want.

In order to write the text lines we just pasted into a file we are going to have to surround them with quotes ("). Since there are already quotes in the musicXML text we will need to escape those quotes or php will get confused between the quotes that are part of the musicXML data and the quotes that are part of the php code. To escape a quote we precede it with a backslash.

```
" -> \"
```

This is very easily done with the search and replace feature of your editor. Just search on " and replace it with \".

Next we need to open the file, bass.musicxml, where we will write our musicXML output. Do that by adding these lines right below the opening tag.

```
// Set the output file
$outputfile = "bass.musicxml";

// Open xml file for output.
$outfile = fopen($outputfile,"w");
```

And when we open a file we have to close a file so add this line above the closing tag.

```
fclose($outfile);
```

Now for the code that actually outputs each line. At the front of each line copied in from the musicXML file add this text.

```
fwrite($outfile, "
```

This can be tricky if you have word wrap on so I recommend you turn word wrap off. Also this is tedious. Later we will devise a better way to do this.

And at the end of each line copied in from the musicXML file add this text.

```
\n");
```

In English this turns each line into a command to write a line to the output file. The part of the line that is actually output is the part between the quotes.

```
fwrite($xmlfile, "This part gets written to the output file\n");
```

You can see why we had to escape the quotes. Otherwise php wouldn't know which quotes are part of the musicXML and which quotes are delimiting the output. When the line is output the backslash is stripped away from the quote leaving us with just a quote in the musicXML file.

And that \n you see near the end of the line stands for newline. Without the \n the output would just be one continuous line of text instead of being laid out line by line in a readable format.

Now php can be just as finicky as musicXML so type the code at the beginning of the first line carefully. Then copy it and paste it to the beginning of the remaining lines.

Same for the 5 character snippet at the end of each line. Type it once carefully and then copy it and paste it to the end of the remaining lines.

At this point you can test the program. Open a terminal or command prompt and type this command.

```
php bass2xml.php
```

Then hit enter.

If this doesn't work correctly you will see php errors. These include the line number with the error. The most common error in php is to forget the semicolon at the end of a line. In that case the error message will probably return the line after the error.

If everything works right the program will output a file, bass.musicxml, that is identical to your exported musicXML file. You can test bass.musicxml by loading it into Finale or any music composition program that has musicXML import.

Soundslice also has a musicXML viewer here.
https://www.soundslice.com/musicxml-viewer/

In Finale if it doesn't play through VST select Midi from the MIDI/Audio menu choice. After I played the imported musicXML file through Midi I switched back to VST and it played fine.

Now that we are outputting valid musicXML we can implement the code replacing the measure blocks. This loop in a loop concept is very common in computer programming and should not be difficult.

Pseudo Code For Writing The MusicXML File

Set the number of measures and open the files.
Set the number of measures to 32.
Open bassline.txt for reading.
Open bassline.musicxml for writing.

Writing the header.
Write the header lines one line at a time up to the first measure block.

Write the measures.
Set the measure number, $measure, to 0.

This is the start of the measure loop.
  Increment (add 1 to) the measure number.

  Read the next four notes from bassline.txt into four variables, $note1, $note2, $note3, $note4 and derive the step, alter, and octave characters.

  Write the measure text up to the note block.

  This is the start of the note loop.

    Write the note block up to the step block.

    Write the step block inserting the first letter from the note variable into the step block.

    Check if the next letter in the note varible is # or b.

    If it is write an alter block including +1 for sharp or -1 for b.

    The note variable ends with the octave number. Write the octave block including the octave number.

    Write the rest of the note block.

  This is the end of the note loop. Since there are 4 notes in a measure loop 4 times total.

  Write the end of the measure block.

this is the end of the measure loop - continue looping until you complete the last measure

Writing the footer
Write the rest of the file one line at a time to the musicXML file.

Indenting Code And Pseudo Code

It is common coding practice to indent code inside a loop and to add additional indentation inside nested loops. This makes it easier to keep track of what code is being executed inside the loops.

Code For Generating The MusicXML File

Working code, bass2xml.php, is included with the book's supplemental materials available at bac.kgpl.org.

Because this is example code I use what is called an unrolled loop. Instead of looping through 4 notes I just unroll the loop and write out the code 4 times, once for each note in the measure.

The reverse engineered musicXML included some data that was neither necessary or correct for generic musicXML output. So the note blocks begin with the text <note> without including the positioning data. I also removed the stem blocks from each note and the transpose block.

When the musicXML file is imported into finale it jams all the bars together on one line so from the Utilities menu choose Fit and set 4 bars per line.

```
<?php
// Set the number of bars
$numberofbars = 32;

// Set the input file
$inputfile = "bassline.txt";

// open input file for reading
$infile = fopen($inputfile,"r");

// Set the output file
$outputfile = "bassline.musicxml";

// Open output file for writing.
$outfile = fopen($outputfile,"w");

// Write the header
fwrite($outfile, "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?>\n");
fwrite($outfile, "<!DOCTYPE score-partwise PUBLIC \"-//Recordare//DTD MusicXML 3.1
Partwise//EN\" \"http://www.musicxml.org/dtds/partwise.dtd\">\n");
fwrite($outfile, "<score-partwise version=\"3.1\">\n");
fwrite($outfile, "  <identification>\n");
fwrite($outfile, "    <rights>©</rights>\n");
fwrite($outfile, "    <encoding>\n");
fwrite($outfile, "      <software>Finale v26.3 for Windows</software>\n");
fwrite($outfile, "      <encoding-date>2021-03-31</encoding-date>\n");
fwrite($outfile, "      <supports attribute=\"new-system\" element=\"print\"
type=\"yes\" value=\"yes\"/>\n");
fwrite($outfile, "      <supports attribute=\"new-page\" element=\"print\"
type=\"yes\" value=\"yes\"/>\n");
fwrite($outfile, "      <supports element=\"accidental\" type=\"yes\"/>\n");
fwrite($outfile, "      <supports element=\"beam\" type=\"yes\"/>\n");
fwrite($outfile, "      <supports element=\"stem\" type=\"yes\"/>\n");
fwrite($outfile, "    </encoding>\n");
```

```
fwrite($outfile, "  </identification>\n");
fwrite($outfile, "  <defaults>\n");
fwrite($outfile, "    <scaling>\n");
fwrite($outfile, "      <millimeters>7.2319</millimeters>\n");
fwrite($outfile, "      <tenths>40</tenths>\n");
fwrite($outfile, "    </scaling>\n");
fwrite($outfile, "    <page-layout>\n");
fwrite($outfile, "      <page-height>1545</page-height>\n");
fwrite($outfile, "      <page-width>1194</page-width>\n");
fwrite($outfile, "      <page-margins type=\"both\">\n");
fwrite($outfile, "        <left-margin>140</left-margin>\n");
fwrite($outfile, "        <right-margin>70</right-margin>\n");
fwrite($outfile, "        <top-margin>70</top-margin>\n");
fwrite($outfile, "        <bottom-margin>70</bottom-margin>\n");
fwrite($outfile, "      </page-margins>\n");
fwrite($outfile, "    </page-layout>\n");
fwrite($outfile, "    <system-layout>\n");
fwrite($outfile, "      <system-margins>\n");
fwrite($outfile, "        <left-margin>0</left-margin>\n");
fwrite($outfile, "        <right-margin>0</right-margin>\n");
fwrite($outfile, "      </system-margins>\n");
fwrite($outfile, "      <system-distance>121</system-distance>\n");
fwrite($outfile, "      <top-system-distance>70</top-system-distance>\n");
fwrite($outfile, "    </system-layout>\n");
fwrite($outfile, "    <appearance>\n");
fwrite($outfile, "      <line-width type=\"stem\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"beam\">5</line-width>\n");
fwrite($outfile, "      <line-width type=\"staff\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"light barline\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"heavy barline\">5</line-width>\n");
fwrite($outfile, "      <line-width type=\"leger\">1.0807</line-width>\n");
fwrite($outfile, "      <line-width type=\"ending\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"wedge\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"enclosure\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"tuplet bracket\">0.918</line-width>\n");
fwrite($outfile, "      <note-size type=\"grace\">60</note-size>\n");
fwrite($outfile, "      <note-size type=\"cue\">60</note-size>\n");
fwrite($outfile, "      <distance type=\"hyphen\">120</distance>\n");
fwrite($outfile, "      <distance type=\"beam\">7.5</distance>\n");
fwrite($outfile, "    </appearance>\n");
fwrite($outfile, "    <music-font font-family=\"Maestro,engraved\" font-
size=\"20.5\"/>\n");
fwrite($outfile, "    <word-font font-family=\"Times New Roman\" font-
size=\"10.25\"/>\n");
fwrite($outfile, "  </defaults>\n");
fwrite($outfile, "  <credit page=\"1\">\n");
fwrite($outfile, "    <credit-type>rights</credit-type>\n");
fwrite($outfile, "    <credit-words default-x=\"632\" default-y=\"53\" font-
size=\"10\" justify=\"center\" valign=\"bottom\">©</credit-words>\n");
fwrite($outfile, "  </credit>\n");
fwrite($outfile, "  <credit page=\"1\">\n");
fwrite($outfile, "    <credit-type>part name</credit-type>\n");
fwrite($outfile, "    <credit-words default-x=\"140\" default-y=\"1478\" font-
size=\"12\" valign=\"top\">Score</credit-words>\n");
fwrite($outfile, "  </credit>\n");
fwrite($outfile, "  <part-list>\n");
fwrite($outfile, "    <score-part id=\"P1\">\n");
```

```
fwrite($outfile, "      <part-name>Piano</part-name>\n");
fwrite($outfile, "      <part-abbreviation>Pno.</part-abbreviation>\n");
fwrite($outfile, "      <score-instrument id=\"P1-I1\">\n");
fwrite($outfile, "        <instrument-name>ARIA Player</instrument-name>\n");
fwrite($outfile, "        <instrument-sound>keyboard.piano</instrument-sound>\n");
fwrite($outfile, "        <virtual-instrument>\n");
fwrite($outfile, "          <virtual-library>Garritan Instruments for
Finale</virtual-library>\n");
fwrite($outfile, "          <virtual-name>005. Keyboards/Steinway Piano</virtual-
name>\n");
fwrite($outfile, "        </virtual-instrument>\n");
fwrite($outfile, "      </score-instrument>\n");
fwrite($outfile, "      <midi-device>ARIA Player</midi-device>\n");
fwrite($outfile, "      <midi-instrument id=\"P1-I1\">\n");
fwrite($outfile, "        <midi-channel>1</midi-channel>\n");
fwrite($outfile, "        <midi-program>1</midi-program>\n");
fwrite($outfile, "        <volume>80</volume>\n");
fwrite($outfile, "        <pan>0</pan>\n");
fwrite($outfile, "      </midi-instrument>\n");
fwrite($outfile, "    </score-part>\n");
fwrite($outfile, "  </part-list>\n");
fwrite($outfile, "  <!--=========================================================--
>\n");
fwrite($outfile, "  <part id=\"P1\">\n");

// Set measure number to 0
$measure = 0;

// Write the measures in a loop
while($measure < $numberofbars){
  // Increment the measure number
  $measure++;

  // read four notes from the file and derive the step, alter, and octave
characters
  $note1 = trim(fgets($infile));
  $note1step = substr($note1,0,1);
  $note1octave = substr($note1, -1);
  $note1alter = "";
  $note1altchar = substr($note1,1,1);
  if($note1altchar == "#"){$note1alter = "+1";}
  if($note1altchar == "b"){$note1alter = "-1";}

  $note2 = trim(fgets($infile));
  $note2step = substr($note2,0,1);
  $note2octave = substr($note2, -1);
  $note2alter = "";
  $note2altchar = substr($note2,1,1);
  if($note2altchar == "#"){$note2alter = "+1";}
  if($note2altchar == "b"){$note2alter = "-1";}

  $note3 = trim(fgets($infile));
  $note3step = substr($note3,0,1);
  $note3octave = substr($note3, -1);
  $note3alter = "";
  $note3altchar = substr($note3,1,1);
  if($note3altchar == "#"){$note3alter = "+1";}
```

```
if($note3altchar == "b"){$note3alter = "-1";}

$note4 = trim(fgets($infile));
$note4step = substr($note4,0,1);
$note4octave = substr($note4, -1);
$note4alter = "";
$note4altchar = substr($note4,1,1);
if($note4altchar == "#"){$note4alter = "+1";}
if($note4altchar == "b"){$note4alter = "-1";}

// Write the measure beginning lines
fwrite($outfile, "    <measure number=\"$measure\" width=\"301\">\n");
fwrite($outfile, "      <print>\n");
fwrite($outfile, "        <system-layout>\n");
fwrite($outfile, "          <system-margins>\n");
fwrite($outfile, "            <left-margin>70</left-margin>\n");
fwrite($outfile, "            <right-margin>0</right-margin>\n");
fwrite($outfile, "          </system-margins>\n");
fwrite($outfile, "          <top-system-distance>177</top-system-distance>\n");
fwrite($outfile, "        </system-layout>\n");
fwrite($outfile, "        <measure-numbering>system</measure-numbering>\n");
fwrite($outfile, "      </print>\n");
fwrite($outfile, "      <attributes>\n");
fwrite($outfile, "        <divisions>2</divisions>\n");
fwrite($outfile, "        <key>\n");
fwrite($outfile, "          <fifths>-1</fifths>\n");
fwrite($outfile, "          <mode>major</mode>\n");
fwrite($outfile, "        </key>\n");
fwrite($outfile, "        <time>\n");
fwrite($outfile, "          <beats>4</beats>\n");
fwrite($outfile, "          <beat-type>4</beat-type>\n");
fwrite($outfile, "        </time>\n");
fwrite($outfile, "        <clef>\n");
fwrite($outfile, "          <sign>F</sign>\n");
fwrite($outfile, "          <line>4</line>\n");
fwrite($outfile, "        </clef>\n");
fwrite($outfile, "      </attributes>\n");
fwrite($outfile, "      <sound tempo=\"120\"/>\n");

// Write the notes
// Note1
fwrite($outfile, "      <note>\n");
fwrite($outfile, "        <pitch>\n");
fwrite($outfile, "          <step>$note1step</step>\n");
if($note1alter == "+1" || $note1alter == "-1"){
  fwrite($outfile,"          <alter>$note1alter</alter>\n");
}
fwrite($outfile, "          <octave>$note1octave</octave>\n");
fwrite($outfile, "        </pitch>\n");
fwrite($outfile, "        <duration>2</duration>\n");
fwrite($outfile, "        <voice>1</voice>\n");
fwrite($outfile, "        <type>quarter</type>\n");
fwrite($outfile, "      </note>\n");

// Note2
fwrite($outfile, "      <note>\n");
fwrite($outfile, "        <pitch>\n");
```

```
  fwrite($outfile, "          <step>$note2step</step>\n");
  if($note2alter == "+1" || $note2alter == "-1"){
    fwrite($outfile,"          <alter>$note2alter</alter>\n");
  }
  fwrite($outfile, "          <octave>$note2octave</octave>\n");
  fwrite($outfile, "        </pitch>\n");
  fwrite($outfile, "        <duration>2</duration>\n");
  fwrite($outfile, "        <voice>1</voice>\n");
  fwrite($outfile, "        <type>quarter</type>\n");
  fwrite($outfile, "      </note>\n");

  // Note3
  fwrite($outfile, "      <note>\n");
  fwrite($outfile, "        <pitch>\n");
  fwrite($outfile, "          <step>$note3step</step>\n");
  if($note3alter == "+1" || $note3alter == "-1"){
    fwrite($outfile,"          <alter>$note3alter</alter>\n");
  }
  fwrite($outfile, "          <octave>$note3octave</octave>\n");
  fwrite($outfile, "        </pitch>\n");
  fwrite($outfile, "        <duration>2</duration>\n");
  fwrite($outfile, "        <voice>1</voice>\n");
  fwrite($outfile, "        <type>quarter</type>\n");
  fwrite($outfile, "      </note>\n");

  // Note4
  fwrite($outfile, "      <note>\n");
  fwrite($outfile, "        <pitch>\n");
  fwrite($outfile, "          <step>$note4step</step>\n");
  if($note4alter == "+1" || $note4alter == "-1"){
    fwrite($outfile,"          <alter>$note4alter</alter>\n");
  }
  fwrite($outfile, "          <octave>$note4octave</octave>\n");
  fwrite($outfile, "        </pitch>\n");
  fwrite($outfile, "        <duration>2</duration>\n");
  fwrite($outfile, "        <voice>1</voice>\n");
  fwrite($outfile, "        <type>quarter</type>\n");
  fwrite($outfile, "      </note>\n");

  // Write the measure ending line
  fwrite($outfile, "    </measure>\n");
}

// write the end of the musicXML file
fwrite($outfile, "  </part>\n");
fwrite($outfile, "  <!--=========================================================--
>\n");
fwrite($outfile, "</score-partwise>\n");

// Close the files
fclose($infile);
fclose($outfile);
?>
```

Copy The Part To The Score

Using the Setup Wizard in Finale make your score with these instruments: Alto Sax, Acoustic Bass, Snare Drum, Bass Drum, and Ride Cymbal. The percussion parts will all be single line instead of a five line staff. Add the title, "Sweet Mint Tea". Change from key of C to key of F and make it 32 bars long.

Then copy the part imported from bassline.musicxml and paste it into the Acoustic Bass line in the score. We will do this with each instrument to complete the piece.

Making It Better

Replacing Repeated Notes

When the bass line is generated using the rules above there are several places where the bass note repeats when the chords go from C7 to Gmin7. On this song this happens at cadential moments which is not terrible but most bass players would not choose to play it that way. A common solution for this is to change the G note on the 4th beat of the C7 to an F encapsulating or surrounding the G.

The program makebass.php described above reads the chord change for "Sweet Mint Tea" and outputs the bass line to bassline.txt. Instead of redoing this work I will write a simple program that reads bassline.txt and outputs basslinemod1.txt with no duplicated notes.

Pseudo Code For Replacing Repeated Notes

```
Open bassline.txt for reading.

Open basslinemod1.txt for writing.

Read note1 and note2 from bassline.txt.

Check if they are the same.

If they are the same change note1 to the note below in the F major scale.

Start the loop until bassline.txt reaches the end of file.

  Write note1 to basslinemod1.txt.

  Set note1 to note2.

  Read note2 from bassline.txt

  Check if note1 and note2 are the same.

  If they are the same change note1 to the note below in the F major scale.

End the loop.

Close the files.
```

Code For Replacing Repeated Notes

Working code, basslinemod1.php, is included with the book's supplemental materials available at bac.kgpl.org.

```
<?php

// Set input file
$inputfile = "bassline.txt";

// Open bassline.txt for reading
$infile = fopen($inputfile, "r");

// Set output file
$outputfile = "basslinemod1.txt";

// Open basslinemod1.txt for writing
$outfile = fopen($outputfile, "w");

// Read the first note
$note1 = trim(fgets($infile));

while(! feof($infile)){
  // Read note2 from th input file
  $note2 = trim(fgets($infile));

  if($note1 == $note2){

    // Change note1
    // Check all notes between F1 and A2
    // If the note is E1 it will stay the same because D1 is outside the range
    switch($note2){
      case "E1":
        break;
      case "F1":
        $note1 = "E1";
        break;
      case "G1":
        $note1 = "F1";
        break;
      case "A1":
        $note1 = "G1";
        break;
      case "Bb1":
        $note1 = "A1";
        break;
      case "C2":
        $note1 = "Bb1";
        break;
      case "D2":
        $note1 = "C2";
        break;
      case "E2":
        $note1 = "D2";
        break;
      case "F2":
        $note1 = "E2";
        break;
      case "G2":
        $note1 = "F2";
        break;
      case "A2":
```

```
        $note1 = "G2";
        break;
      default:
        echo "Note not valid.";
    } // end switch
  } // end if
  // Write note1 to the output file
  fwrite($outfile, $note1."\n");

  // Set note1 to note2
  $note1 = $note2;
} //end while

// Write thelast note to the output file
//fwrite($outfile, $note1."\n");

// Close the files
fclose($infile);
fclose($outfile);
?>
```

After you have run this program you will have basslinemod1.txt without the repeated notes. Edit bass2xml.php to change the input and output files by adding the mod1 suffix to the file names before the extension. Run bass2xml.php to create basslinemod1.musicxml. Check it out. I prefer this version so I pasted it into the score.

Further Exploration

Change the encapsulation avoiding the repeated notes to encapsulate chromatically instead of diatonically. Instead of choosing the note in the F major scale below the target note choose the note a half step below the target note. The changes to the code, baselinemod1.php, are minimal but you will also have to modify bass2xml.php to add the notes that are not in the F major scale.

Add some more chords that are not in the church modes. V7 of V is commonly used and would be a good starting place to support more complex harmonies. The diminished chords and V7b9 would also be good chords to add.

The algorithm above is deterministic. That means for every input file the exact same bass line will be generated. And it also means that every 32 bars will have the exact same part. By adding some randomization into the algorithm the output will not always be exactly the same.

One simple way to add randomization to the algorithm is to make a random selection when there is a choice of root notes. Sometimes jump the shortest distance and sometimes jump the farthest distance. There is a rand() function in php to accomplish this. We will be using randomization in the Melody Section of this book.

Walking bass is only one style for jazz bass. Other common styles include Latin and Funk. Write about or discuss how to implement other styles. Write pseudo code and/or code for other styles. Research other genres of music like rock, blues, or reggae to learn how to auto generate bass parts for these styles.

Write about or discuss possible bass lines for music not in the traditional genres. Devise bass lines for your own brand new style. Implement these bass parts in pseudo code and/or code.

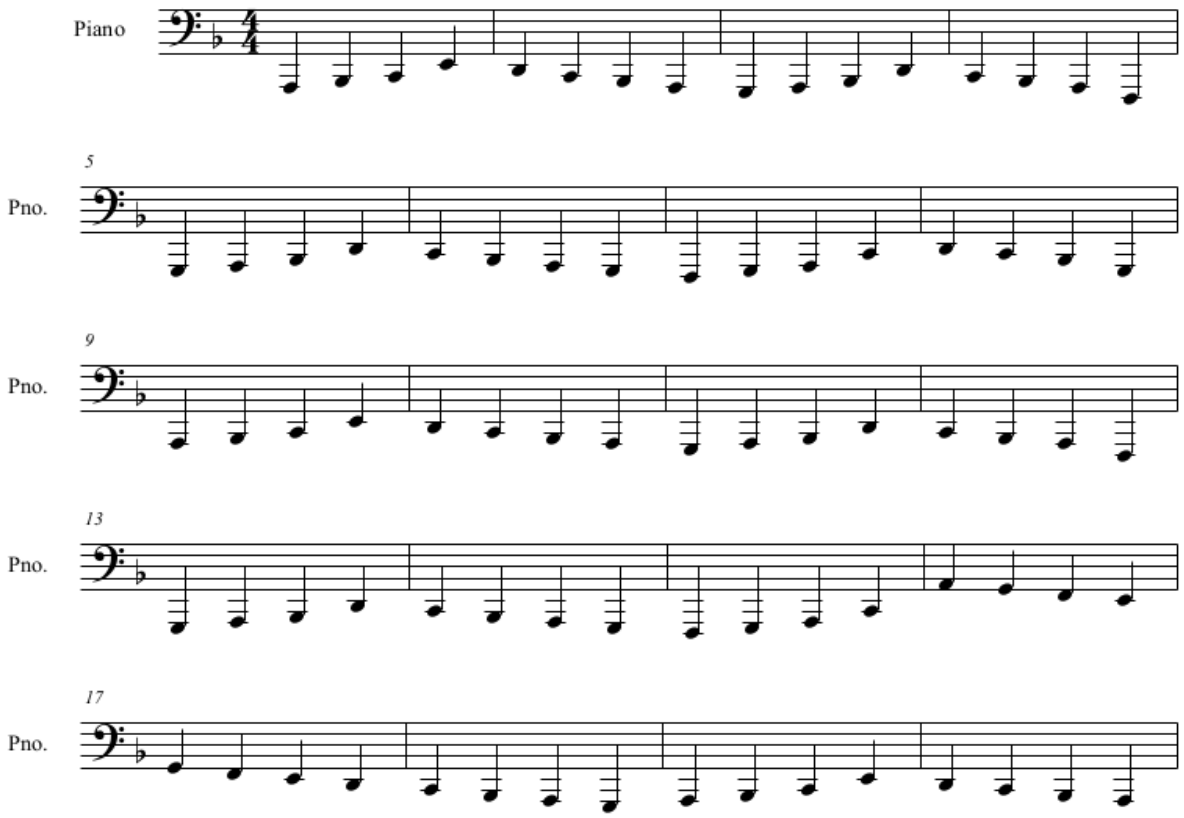
Some Real World Examples

There are commercial applications that use algorithmic composition to write bass lines and other parts. These are sold as educational tools for playing songs while you rehearse them and as backing tracks for musicians to use to play or sing over. Two of these applications are Band-in-a-Box and iREAL PRO. There are many others. It is worthwhile to listen to these apps generate parts for jazz songs. Like this example program they take a chord change as input and then play a bass part that works over that chord change. They are much more sophisticated than the simple algorithm we have used to generate a bass part.

These applications are also useful in music composition. Composers can input their own chord changes and then play them in a loop while searching for melodies on their instrument. As good as they are they still don't play like a real musician. Composer's benefit from readings of their work but musicians are not available anytime like an application is.

Band-in-a-Box - https://www.pgmusic.com/

iREAL PRO - https://www.irealpro.com/



Bass line without repeated notes imported from musicXML file.

Notice that this bass part is in concert pitch. Bass is played an octave below where written to avoid excessive staff lines below the bass clef.

Using Code To Write Code

Converting a musicXML file into code that outputs the same musicXML file as described above is tedious and error prone. Just like programmers write code to automate the output of musicXML they also write code to output program files. This is especially effective for tedious, error prone jobs that are not complex in any way.

If you want to output your algorithmic compositions to musicXML with reverse engineered templates as described above using a program like this will save on a lot of copying and pasting and will give you error free results. The procedure is as follows.

Enter a bar or two of music in Finale or another notation program that can export musicXML. Include anything in that file that you want to include in your output, title, instrument, composer, dynamics, articulation, etc.

Export that to a musicXML file. Be sure to select the .musicxml extension, not the default .mxl extension. The .mxl files are compressed, not plain text, and they won't work for this program.

The exported file then becomes the input for this program that converts it to a php program.

Then you replace the measure blocks with the measures and notes that are derived by your algorithm. The measure blocks also serve as a template to use inside your loops providing the correct musicXML syntax and the right places to input your note values for your generated music.

Writing a program that writes another program is tricky and confusing. Although it is a technique that can be used for algorithmic composition it is an advanced technique and I do not recommend it for beginning programmers.

I am including the program here for you to use. It will make it easier for you to work with musicXML templates and turn them into code that can then be modified to implement your algorithm.

Pseudo Code For Converting MusicXML To A Program That Outputs MusicXML

This code works with any text file, not just XML or musicXML files. When you have a text file template for your output you can run this program on the template and then add loops and insert data from variables or databases in the appropriate places.

So first, this program writes a php program that will output the template. Then the programmer can modify that program to produce output based on the template but with the changes made where required.

Here's The Pseudo Code

Set variables for input, output, and program files.

Initialize the prefix and suffix lines that turn the text lines into php.

Open the input file for reading.

Open the program file for writing.

Write the php header.

Output the php code that opens the output file for writing.

Output the php code that starts the read file loop.

Input a line, trim it on the right, and escape the quotes.

Surround the line with prefix in front then newline and suffix at the end.

Output the code that writes the line to the output file.

End the loop.

Close the output musicXML file in the code and write the ending tag.

Close the input and output files.

Code That Converts A MusicXML File Into A PHP Program That Outputs The MusicXML Back To A File

Working code, xml2php.php, is included with the book's supplemental materials available at bac.kgpl.org.

This code is tricky and is included as a tool for reverse engineering musicXML files. If you want to examine or modify this code it is included in the supplemental materials.

In order to use the code copy it into a folder with the xml template you have exported from Finale. Then change the three variables at the top called $inputfile, $outputphp, and $outputfile. Open a terminal or command prompt and type this command.

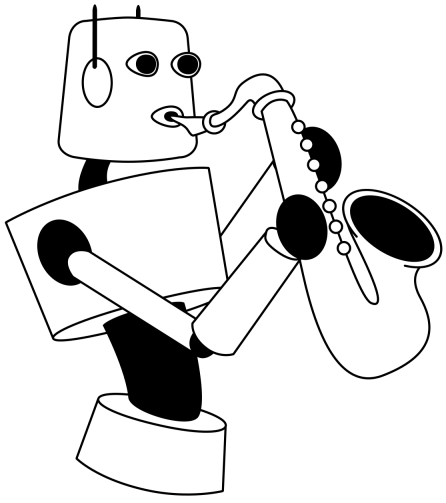
php xml2php.php

Then hit enter.

This will produce your output php program file. Run that file and you will get your output xml file identical to your input musicxml file. Then change the code in the output php file adding loops and variable names as needed to implement your algorithm.

Since this is a utility, not an algorithmic composition program I don't include the code in this book. You will find it in the xml2txt folder in the supplemental materials.

The robots and the cover art were drawn by Kier Heyl.



Melody

I have a melody generating algorithm that uses a database of notes and some random selection. I use the Charlie Parker Omnibook as a reference to build a database based on the notes Parker plays in "Confirmation". Any sufficiently long and varied piece of music could be used or the composer can devise their own notes providing another creative input to the algorithm.

Here is an explanation of the algorithm.
I list all three note sequences in the reference piece like this.

```
note1, note2, note3
note2, note3, note4
.
.
.
noteN-1, noteN, note1
noteN, note1, note2
```

Notice how the list of note sequences wraps around from the end of the composition back to the beginning. This is necessary or it is possible that the algorithm will fail when trying to find a possible third note to go after the last two notes if they are selected. Say the last two notes are G and C and that this is the only time in the piece where there is a G followed by a C. If we don't wrap around to the first note there will be no third note that follows G and C breaking the algorithm.

So to see how this will actually look let's say the first four notes are Bb3 D4 C4 Eb4 and lets say the last two notes are Bb4 C5.

```
Bb3, D4, C4
D4, C4, Eb4
.
.
.
Bb4, C5, Bb3
C5, Bb3, D4
```

This notation corresponds exactly to the easier to understand notation above using note1, note2, ...

Now I select any one of these sequences to start the melody. After writing these notes down I want to write the next note. I know the two notes that the next note follows because they are the last two I have written down. I select all the sequences that start with those two notes and I choose randomly between them to get the next note. There will always be at least one sequence that matches because those two notes came from the original melody or data set that the sequences are derived from.

Rinse wash and repeat. Look at the last two notes of the melody being generated. Select all sequences that start with those two notes. Pick randomly from the selected sequences to choose the next note. And continue until the melody is long enough, time runs out, or forever. (Which raises a question - can the algorithm continue running after the heat death of the universe?)

So the algorithm in a nutshell is first prepare the database from the source material as described. Select the starting notes randomly from the data set. Then

enter an infinite loop picking the next note randomly from all sequences where the first two notes match the last two notes in the generated melody.

Assuming a sufficiently large and varied data set the generated melody remains new. If the note data set is not large enough or varied enough the resulting melodies may well be constrained. For instance if there is only one sequence matching the two starting notes for every possible set of starting notes the random element will be removed and the melody will repeat verbatim. Or if there is only one sequence for most of the two note pairs there may be frequently repeated melodic motifs.

How long and varied must the data set be to deliver desirable results? That is a choice for the composer to make. On some pieces a small data set may produce satisfying results. On others the melody may seem repetitive and a larger input data set will be required.

How To Create A Database For The Algorithm

For many good reasons most computer databases are created using a Database Management System or DBMS. A good DBMS will support very large data files, a variety of data types, a query system like SQL (Structured Query Language), and indexing to make queries fast and efficient. None of this applies to this project so for this example I am going to use a text database instead.

The reasons to use a text database are both the ease of reading and editing text files and so we don't have to learn a query language but can instead do the queries using php.

For projects with larger data sets I can recommend mysql or mariadb. Libraries are included in php and most programming languages to make it easy to insert the data and make queries. Since this project will use text files I won't go any further into how to use a DBMS except to include links in the glossary.

Creating The Database For This Example Takes Several Steps

1. Select or create the source material.
2. Make a list of all the notes in order.
3. Create another list of all three note sequences in the note list including the wraparound at the end.
4. Build a database from the list of sequences that can be queried by the first two notes in the sequence.

The Melody Input File

First I listed all the notes in the source material in a text file called melodyin.txt. The first note went on the first line, the second note on the second line, and so on. Then to achieve the wraparound described above I copied the first two lines to the bottom of the text file.

Typing the notes in is tedious and exacting. I recommend delineating small sections with white space or a row of dashes to make proofreading easy. I also recommend end of page notation like eop. Write the dashes or eop on their own lines so they will be easy to delete.

Another minor problem is enharmonic note spelling. All of the black notes on a piano have two names, C# is the same note as Db. In "Confirmation" the transcriber used both Gb and F#. A quick search and replace in the text editor can change all the F#s to Gb. If you don't do this and some of the sequences start with Gb while

others start with F# the random choices for the next note will be smaller than they could be skewing the probability of that choice.

In the end you will have a file that looks like melodyin.txt which is included in the supplemental materials. Here are the first seven lines of that file.

```
A4
C5
A4
Bb4
A4
E4
F4
.
.
```

The Sequences File

The code in makeseq.php takes melodyin.txt as the input file and outputs sequences.txt as the output file. One difference between the sequences.txt output file and the examples above is that it will end up in sorted order. This will make it easier to generate the final text database as described below.

I will use arrays in the makeseq.php code. An array is a structured list just like a text file except that the array is present only when the program is running and the text file output will remain available on the system as sequences.txt. Another minor but important detail is that arrays usually start with element 0 instead of element 1 while the first line in a text file is normally called line 1. Here's a visualization of the array built from the melodyin.txt input file.

```
Array[0] = A4
Array[1] = C5
Array[2] = A4
Array[3] = Bb4
Array[4] = A4
.
.
```

Pseudo Code For Listing The Sequences

Read the input file into a notes array, $notesarray[].

Skip lines that don't start with A through G. Skipping these lines avoids problems with extra white space or delimiters left in the file and so only includes valid note data.

The number of starting notes will be the length of the array -2 because of the wrap around notes added to the bottom. Here's where the fact that arrays start with element 0 becomes tricky. We will call the number of starting notes $notes.

Loop through the array like this.

```
$notes = number of starting notes
$index = 0

while($index > $notes) // Again tricky here. Might be >=.
  $line = Array[index]."-".Array[index+1]."-".Array[index+2] // Add dashes between
notes
```

```
  Write $line to sequence array.
  $sequencearray[$index] = $line

  Increment $index. // Add one to the index value.

Loop back to while statement.

Sort sequence array.

Write the sequence array to the output file.
```

Code For Making The Sequence Files

Working code for makeseq.php is included with the book's supplemental materials available at bac.kgpl.org. To run this program copy it to the same folder with your melody input file, in this example melodyin.txt. Then open a terminal or command prompt and type this command.

```
php makeseq.php
```

Then hit enter.

Control the input and output filenames by changing the variables at the top of the file.

```
<?php

// Set input file
$inputfile = "melodyin.txt";

// Open the melody input file
$infile = fopen($inputfile,"r");

// Set output file
$outputfile = "data1.txt";

// Open output file  for writing
$outfile = fopen($outputfile,"w");

// set the variable for the array index
$index = 0;

// Read the lines one at a time in a while loop
while(! feof($infile)){

  // Read the chord from the input file
  $chord = trim(fgets($infile));

  // Get the first letter in the chord
  $fl = substr($chord,0,1);

  // Check if the character is between A and G
  // The pipes, ||, are read as OR.
  if ($fl=="A"||$fl=="B"||$fl=="C"||$fl=="D"||$fl=="E"||$fl=="F"||$fl=="G"){

    // Add the chord to the notes array
    $notesarray[$index] = $chord;
```

```
    // Increment the index
    $index++;
  }
}

// Close the input file melodyin.txt
fclose($infile);

// Test line to display notes array.
//print_r($notesarray);

// Test line to display the value of $index
//echo $index;

// Set the number of notes less the last two for the wraparound
$notes = $index-2;

// Reset the index
$index = 0;

// Start the loop to output the three note sequences
while($index < $notes){

  // Read a three note sequence into a line with dashes
  $line = $notesarray[$index]."-".$notesarray[$index+1]."-".$notesarray[$index+2];

  // Write the line to the sequences array, sarray[].
  $sarray[$index] = $line;

  // Increment the index to start the next sequence
  $index++;
}

// Sort the sequences array.
sort($sarray);

// Write the sequence out to the output file with a newline character
// Reset the index to 0.
$index = 0;
while($index < $notes){
  // Write the next sequence to the output file
  fwrite($outfile, $sarray[$index]."\n");

  // Increment the index
  $index++;
}

// Close the output file
fclose($outfile);

?>
```

Create The Search Database

In creating a text database I will use the names of the text files to indicate the last two notes played and use the contents of the file to store the choices for the next note. The first line of the text file will indicate the total number choices to select from.

So, say the last two notes played were C4 and Eb4. And let's say there are three sequences that start with C4 Eb4 ending in C4, F4, and G4.

We need to create a text file called C4-Eb4.txt and the contents of the file need to be

```
3
C4
F4
G4
```

When I am outputting the melody I know the last two notes are C4 and Eb4 because I just wrote them. So I open the file called C4-Eb4.txt and I read the first line, 3. Then I use the random function to randomly select 1, 2, or 3. If I get 1 the next note will be C4. If I get 2 the next note will be F4. And if I get 3 the next note will be G4.

It is often the case that there are identical three note sequences played at different places in the source material. So we could have a file like this.

```
4
C4
C4
F4
G4
```

If this happens we can see that the probability of selecting C4 for the next note is one chance in two while the probability of selecting F4 or G4 are each one chance in 4. This reflects the occurrence of these sequences in the source material.

So now we have to output a set of files using sequences.txt as the input. We will call this program makedatabase.php. Since sequences.txt is sorted all sequences starting with the same notes will be grouped together like this.

```
A3-A4-Bb4
A3-A4-C5
A3-Bb3-B3
A3-Bb3-D4
A3-C4-A3
A3-C4-A4
A3-C4-A4
A3-C4-A4
.
.
```

Pseudo Code For Making The Search Database

```
Set the data directory.

Open the input file.

Read the first line from the input file into a variable $currentsequence.

Extract the variables $note1, $note2, $note3 from $currentsequence.

Set a variable, $twonotes, to $note1-$note2.

Set another variable, $fname, to note1-note2.txt.

Set a variable, $counter, to 0.

Start nested loops like this.

Loop through the input file until all sequences are read.
while(not end of file)
  Loop through all the lines that match $twonotes.
  while($twonotes is the same as $note1-$note2)
    Write the last note to an array indexed by the counter.
    notesout[$counter] = note3

    Increment the counter.
    $counter = $counter+1

    Read the next line from the input file into the variable $currentsequence.

    Extract the variables $note1, $note2, $note3 from $currentsequence.
  end the nested while loop.

This loop should terminate whenever we are done with all the sequences that start
with $twonotes which was derived using $note1 and $note2 from our starting
sequence. In other words we continue until the first two notes don't match. Since
the input file is sorted that will be all of the sequences starting with those two
notes.

  Then we write the file, $fname like this.

  The first line gets $counter.

  Set an index to 0 and loop.
  While($index < $counter)
    Increment $index.
    The next line gets $notesout[$index].
  End nested while loop for writing to the database.
End while loop for input file.

This will result in a text database with one file for each unique two note starting
sequence. Each of these files will hold the number of matches for that sequence on
the first line and will list the third note in the sequence on the subsequent
lines. These files and the output file are the text database that will allow us to
generate a melody using the algorithm described above.
```

Code For Making The Search Database

Working code for makedatabase.php is included with the book's supplemental materials available at bac.kgpl.org. Run makedatabase.php in the same folder as data1.txt. Make a subfolder called data1 in that folder. This is where the text database will be written. If you are using different source material you can send it to data2.txt and write the text database to the data2 folder by changing the values of the variables at the top of makeseq.php and makedatabase.php.

```
<?php

// Set data directory
$datadir = "data1/";

// Set input file
$inputfile = "data1.txt";

// open sequences.txt to read
$infile = fopen($inputfile,"r");

// Read the first three note sequence from sequences.txt to $currentsequence
$currentsequence = fgets($infile);

// Extract the variables $note1, $note2, $note3 from $currentsequence.
// note1 goes into chordarray[0], note2 into chordarray[1], and note3 into
chordarray[2]
$chordarray = explode("-",$currentsequence);

// Loop through the inpute file
while(! feof($infile)) {

  // Read the first two notes in the sequence into varibale, $twonotes.
  $twonotes = $chordarray[0]."-".$chordarray[1];

  // Build the output filename for these two notes.
  $fname = $twonotes.".txt";

  // Set counter to 0
  $counter = 0;

  // Initialize the $notesout[] array to empty
  $notesout = [];

  // This loop gathers all lines starting with note1 and note2 into an array
  while($twonotes == $chordarray[0]."-".$chordarray[1]){
    // Write the third note to an array, $notesout[], indexed by $counter.
    $notesout[$counter] = $chordarray[2];

    // Increment the counter.
    $counter++;

    //Read the next line from sequences.txt into the variable $currentsequence
    $currentsequence = fgets($infile);

    // Check to see if sequences.txt is at end of file.
    if(feof($infile)){
      // When we empty the input file, sequences.txt, break out of the while loop.
```

```
      break;
    }

    // Extract the variables $note1, $note2, $note3 from $currentsequence.
    $chordarray = explode("-",$currentsequence);
  }

  // Open the output file for these two notes
  $outfile = fopen($datadir.$fname,"w");

  // Write the counter to the first line
  fwrite($outfile,$counter."\n");

  // Initialize the index to 0
  $index = 0;

  // Loop through the found third notes
  while($index < $counter){
    // Write the third note in the sequence to the next line of the output file
    fwrite($outfile,$notesout[$index]);

    // Increment the output file
    $index++;
  }
  // Close this output file
  fclose($outfile);
}

// Close input file
fclose($infile);

?>
```

Generating The Melody

Now that we have the text database we can generate the melody. Here's a quick review of the algorithm and how it works with the text database.

Pick a random three note sequence from sequences.txt to start the melody.

Find all instances of the last two notes written and then randomly choose from the notes following them. These will be in the file note1-note2.txt.

Continue ad infinitum (or until it's long enough). The text database built in the previous steps is designed to make it easy to implement this algorithm.

Pseudo Code For The Melody Algorithm

Pick a random three note sequence from the input file. Output these three notes, one note per line, to the melody file.

Remember the last two notes written as $lastnote1 and $lastnote2.

This is the start of the loop.

```
while(not long enough)
```

```
  Set output file to $lastnote1-$lastnote2.txt.

  Set $lastnote1 to $lastnote2 for the next iteration of the loop.
  $lastnote1 = $lastnote2

  Open the file $lastnote1-$lastnote2.txt and read the first line, a number.

  Choose randomly another number from 1 to the number just read. Call it $n.

  Read one line at a time $n times.

  Set $lastnote2 to the line just read.

  Output $lastnote2 to to melodyout.txt.

End while loop.
```

Code For The Melody

Working code, makemelody.php, is included with the book's supplemental materials available at bac.kgpl.org. Run makemelody.php in the same folder as the data folder, data1 in this example. Then open a terminal or command prompt and type this command.

```
php makemelody.php
```

Then hit enter.

Set the input file, the database folder, and the output file by changing the variables at the top of the program.

```
<?php

// Set data directory
$datadir = "data1/";

// Set sequences input file
$inputfile = "data1.txt";

// Open the sequenses input file
$infile = fopen($inputfile,"r");

// Set output file
$outputfile = "melodyout.txt";

// Open output file  for writing
$outfile = fopen($outputfile,"w");

// Set $count to 0.
$count = 0;

// Loop through the file one line at a time.
while(fgets($infile)){
  // Increment $count.
  $count++;
}
```

```
// Close the file sequences.txt.
fclose($infile);

// Pick a random number from 1 to $count;
$randomnumber = rand(1, $count);

// Get the sequence selected by $randomnumber.
// Open the input file again to read from the top
$infile = fopen($inputfile,"r");

// Set $index to 0.
$index = 0;

// Loop through the file one line at a time.
while($index < $randomnumber){

  // Read the next sequence and strip whitespace and newline.
  $sequence = trim(fgets($infile));

  // Increment $index.
  $index++;
}

// Close the file sequences.txt.
fclose($infile);

// Extract the variables $note1, $note2, $note3 from $sequence.
$chordarray = explode("-",$sequence);

// Build the text file name from the selected sequence.
$twonotefile = $chordarray[1]."-".$chordarray[2].".txt";

// Remember then last note. It will be needed in the loop.
$lastnote2 = $chordarray[2];

// Set $index to 0. This index will terminate the loop.
$index = 0;

// Set $barnum to the number of bars to be output.
$barnum = 32;

// Compute $notenum to the number of notes to be output.
$notenum = 8 * $barnum;

// Loop to write the melody.
// The number in this line sets the number of notes output.
while($index < $notenum){

  // Open the two note database file for read.
  $datafile = fopen($datadir.$twonotefile,"r");

  // Set $lastnote1 to $lastnote2, ready for the next iteration of the loop.
  $lastnote1 = $lastnote2;

  // Read the first line of the file, a number.
  $number = trim(fgets($datafile));
```

```
  // Pick a random number from 1 to $number;
  $randomnumber = rand(1, $number);

  // Set another index to $seqnum.
  $seqnum = 0;

  // Loop until we get to the randomly selected sequence.
  while($seqnum < $randomnumber){
    // Read the last note.
    $lastnote2 = trim(fgets($datafile));

    // Increment $seqnum
    $seqnum++;
  }

  // Write the note to the output file.
  fwrite($outfile,$lastnote2."\n");

  // Build the next data file name from lastnote1 and lastnote2.
  $twonotefile = $lastnote1."-".$lastnote2.".txt";

  // Close the data file.
  fclose($datafile);

  // Increment the index.
  $index++;
}

// Close the output file.
fclose($outfile);

?>
```

Generating The MusicXML File

This is done just like we did for the bass part. One difference is that the melody is made up of eighth notes where the bass part has only quarter notes. Another difference is that the notes loop is not unrolled. Instead we use parallel arrays to keep up with the step, octave, and alter values. So we have arrays for note, notestep, octave, etc. all written by the same index and then, later, read by the same index.

Pseudo Code For The Melody Algorithm

Set the number of measures and open the files.
Set the number of measures to 32.
Open the input file created by makemelody.php for reading.
Open the output file for writing.

Write the header.
Write the header lines one line at a time up to the first measure block.

Write the measures.
Set the measure number, $measure, to 0.

This is the start of the measure loop.
  Increment (add 1 to) the measure number.

Use a for loop to read the next 8 notes and derive the step, alter, and octave characters. Save these values into parallel arrays so we have $note[1], $notestep[1], $noteoctave[1], $notealter[1], and $notealtchar[1] for the first note and so on for the rest of the 8 notes in the measure.

Write the measure text up to the note block.

This is the start of the note loop.
Write the note block up to the step block.

Write the step block inserting the first letter from the note variable into the step block.

Check if the next letter in the note varible is # or b.

If it is write an alter block including +1 for sharp or -1 for b.

Write the octave block including the octave number.

Write the rest of the note block.

This is the end of the note loop - since there are 8 notes in a measure loop 8 times.

Write the end of the measure block.

This is the end of the measure loop. Continue looping until you complete the last measure.

Writing the footer
Write the rest of the file one line at a time to complete the musicXML file.

Code For Writing The MusicXML File For The Melody

Working code, melody2xml.php, is included with the book's supplemental materials available at bac.kgpl.org. To run this code copy melody2xml.php into the same folder as the melodyout.txt file generated by makemelody.php as described above. Then open a terminal or command prompt and type this command.

```
php melody2xml.php
```

Then hit enter.

Set the number of bars, the input file, and the output file by changing the variables at the top of the program.

```
<?php

// Set the number of bars
$numberofbars = 32;

// Set the input file
$inputfile = "melodyout.txt";

// open input file for reading
$infile = fopen($inputfile,"r");
```

```
// Set the output file
$outputfile = "melody.musicxml";

// Open xml file for output.
$outfile = fopen($outputfile,"w");

// Output the header lines.
fwrite($outfile, "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?>\n");
fwrite($outfile, "<!DOCTYPE score-partwise PUBLIC \"-//Recordare//DTD MusicXML 3.1
Partwise//EN\" \"http://www.musicxml.org/dtds/partwise.dtd\">\n");
fwrite($outfile, "<score-partwise version=\"3.1\">\n");

// Use the input file name as the title
fwrite($outfile, "  <movement-title>$outputfile</movement-title>\n");

fwrite($outfile, "  <identification>\n");
fwrite($outfile, "    <creator type=\"composer\">Larry Heyl</creator>\n");
fwrite($outfile, "    <rights>© by Larry Heyl, 2021</rights>\n");
fwrite($outfile, "    <encoding>\n");
fwrite($outfile, "      <software>Finale v26.3 for Windows</software>\n");
fwrite($outfile, "      <encoding-date>2021-04-01</encoding-date>\n");
fwrite($outfile, "      <supports attribute=\"new-system\" element=\"print\"
type=\"yes\" value=\"yes\"/>\n");
fwrite($outfile, "      <supports attribute=\"new-page\" element=\"print\"
type=\"yes\" value=\"yes\"/>\n");
fwrite($outfile, "      <supports element=\"accidental\" type=\"yes\"/>\n");
fwrite($outfile, "      <supports element=\"beam\" type=\"yes\"/>\n");
fwrite($outfile, "      <supports element=\"stem\" type=\"yes\"/>\n");
fwrite($outfile, "    </encoding>\n");
fwrite($outfile, "  </identification>\n");
fwrite($outfile, "  <defaults>\n");
fwrite($outfile, "    <scaling>\n");
fwrite($outfile, "      <millimeters>7.2319</millimeters>\n");
fwrite($outfile, "      <tenths>40</tenths>\n");
fwrite($outfile, "    </scaling>\n");
fwrite($outfile, "    <page-layout>\n");
fwrite($outfile, "      <page-height>1545</page-height>\n");
fwrite($outfile, "      <page-width>1194</page-width>\n");
fwrite($outfile, "      <page-margins type=\"both\">\n");
fwrite($outfile, "        <left-margin>140</left-margin>\n");
fwrite($outfile, "        <right-margin>70</right-margin>\n");
fwrite($outfile, "        <top-margin>70</top-margin>\n");
fwrite($outfile, "        <bottom-margin>70</bottom-margin>\n");
fwrite($outfile, "      </page-margins>\n");
fwrite($outfile, "    </page-layout>\n");
fwrite($outfile, "    <system-layout>\n");
fwrite($outfile, "      <system-margins>\n");
fwrite($outfile, "        <left-margin>0</left-margin>\n");
fwrite($outfile, "        <right-margin>0</right-margin>\n");
fwrite($outfile, "      </system-margins>\n");
fwrite($outfile, "      <system-distance>121</system-distance>\n");
fwrite($outfile, "      <top-system-distance>70</top-system-distance>\n");
fwrite($outfile, "    </system-layout>\n");
fwrite($outfile, "    <appearance>\n");
fwrite($outfile, "      <line-width type=\"stem\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"beam\">5</line-width>\n");
fwrite($outfile, "      <line-width type=\"staff\">0.918</line-width>\n");
```

```
fwrite($outfile, "      <line-width type=\"light barline\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"heavy barline\">5</line-width>\n");
fwrite($outfile, "      <line-width type=\"leger\">1.0807</line-width>\n");
fwrite($outfile, "      <line-width type=\"ending\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"wedge\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"enclosure\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"tuplet bracket\">0.918</line-width>\n");
fwrite($outfile, "      <note-size type=\"grace\">60</note-size>\n");
fwrite($outfile, "      <note-size type=\"cue\">60</note-size>\n");
fwrite($outfile, "      <distance type=\"hyphen\">120</distance>\n");
fwrite($outfile, "      <distance type=\"beam\">7.5</distance>\n");
fwrite($outfile, "    </appearance>\n");
fwrite($outfile, "    <music-font font-family=\"Maestro,engraved\" font-
size=\"20.5\"/>\n");
fwrite($outfile, "    <word-font font-family=\"Times New Roman\" font-
size=\"10.25\"/>\n");
fwrite($outfile, "  </defaults>\n");
fwrite($outfile, "  <credit page=\"1\">\n");
fwrite($outfile, "    <credit-type>title</credit-type>\n");
fwrite($outfile, "    <credit-words default-x=\"632\" default-y=\"1475\" font-
size=\"24\" justify=\"center\" valign=\"top\">Melody</credit-words>\n");
fwrite($outfile, "  </credit>\n");
fwrite($outfile, "  <credit page=\"1\">\n");
fwrite($outfile, "    <credit-type>composer</credit-type>\n");
fwrite($outfile, "    <credit-words default-x=\"1122\" default-y=\"1407\" font-
size=\"12\" justify=\"right\" valign=\"top\">Larry Heyl</credit-words>\n");
fwrite($outfile, "  </credit>\n");
fwrite($outfile, "  <credit page=\"1\">\n");
fwrite($outfile, "    <credit-type>rights</credit-type>\n");
fwrite($outfile, "    <credit-words default-x=\"632\" default-y=\"53\" font-
size=\"10\" justify=\"center\" valign=\"bottom\">© by Larry Heyl, 2021</credit-
words>\n");
fwrite($outfile, "  </credit>\n");
fwrite($outfile, "  <credit page=\"1\">\n");
fwrite($outfile, "    <credit-type>part name</credit-type>\n");
fwrite($outfile, "    <credit-words default-x=\"140\" default-y=\"1478\" font-
size=\"12\" valign=\"top\">Score</credit-words>\n");
fwrite($outfile, "  </credit>\n");
fwrite($outfile, "  <part-list>\n");
fwrite($outfile, "    <score-part id=\"P1\">\n");
fwrite($outfile, "      <part-name>Piano</part-name>\n");
fwrite($outfile, "      <part-abbreviation>Pno.</part-abbreviation>\n");
fwrite($outfile, "      <score-instrument id=\"P1-I1\">\n");
fwrite($outfile, "        <instrument-name>ARIA Player</instrument-name>\n");
fwrite($outfile, "        <instrument-sound>keyboard.piano</instrument-sound>\n");
fwrite($outfile, "        <virtual-instrument>\n");
fwrite($outfile, "          <virtual-library>Garritan Instruments for
Finale</virtual-library>\n");
fwrite($outfile, "          <virtual-name>005. Keyboards/Steinway Piano</virtual-
name>\n");
fwrite($outfile, "        </virtual-instrument>\n");
fwrite($outfile, "      </score-instrument>\n");
fwrite($outfile, "      <midi-device>ARIA Player</midi-device>\n");
fwrite($outfile, "      <midi-instrument id=\"P1-I1\">\n");
fwrite($outfile, "        <midi-channel>1</midi-channel>\n");
fwrite($outfile, "        <midi-program>1</midi-program>\n");
fwrite($outfile, "        <volume>80</volume>\n");
```

```
fwrite($outfile, "        <pan>0</pan>\n");
fwrite($outfile, "      </midi-instrument>\n");
fwrite($outfile, "    </score-part>\n");
fwrite($outfile, "  </part-list>\n");
fwrite($outfile, "  <!--=========================================================--
>\n");
fwrite($outfile, "  <part id=\"P1\">\n");

// Next we will loop through the measures reading the notes from inputfile and
writing one measure at a time.
$measure = 0;

while($measure < $numberofbars ){
  $measure++;

  // read eighth notes from the file in a for loop and derive the step, alter, and
octave characters
  // each variable has it's own array taking values 1 through 8 for the 8 notes in
the measure
  for ($index = 1; $index <= 8; $index++){
    $note[$index] = trim(fgets($infile));
    $notestep[$index] = substr($note[$index],0,1);
    $noteoctave[$index] = substr($note[$index], -1);
    $notealter[$index] = "";
    $notealtchar[$index] = substr($note[$index],1,1);
    if($notealtchar[$index] == "#"){$note1alter = "+1";}
    if($notealtchar[$index] == "b"){$note1alter = "-1";}
  }

  // write the musicXML for this measure
  fwrite($outfile, "    <measure number=\"$measure\">\n");
  fwrite($outfile, "      <print>\n");
  fwrite($outfile, "        <system-layout>\n");
  fwrite($outfile, "          <system-margins>\n");
  fwrite($outfile, "            <left-margin>70</left-margin>\n");
  fwrite($outfile, "            <right-margin>0</right-margin>\n");
  fwrite($outfile, "          </system-margins>\n");
  fwrite($outfile, "          <top-system-distance>177</top-system-distance>\n");
  fwrite($outfile, "        </system-layout>\n");
  fwrite($outfile, "        <measure-numbering>system</measure-numbering>\n");
  fwrite($outfile, "      </print>\n");
  fwrite($outfile, "      <attributes>\n");
  fwrite($outfile, "        <divisions>2</divisions>\n");
  fwrite($outfile, "        <key>\n");
  fwrite($outfile, "          <fifths>-1</fifths>\n");
  fwrite($outfile, "          <mode>major</mode>\n");
  fwrite($outfile, "        </key>\n");
  fwrite($outfile, "        <time>\n");
  fwrite($outfile, "          <beats>4</beats>\n");
  fwrite($outfile, "          <beat-type>4</beat-type>\n");
  fwrite($outfile, "        </time>\n");
  fwrite($outfile, "        <clef>\n");
  fwrite($outfile, "          <sign>G</sign>\n");
  fwrite($outfile, "          <line>2</line>\n");
  fwrite($outfile, "        </clef>\n");
  fwrite($outfile, "      </attributes>\n");
  fwrite($outfile, "      <sound tempo=\"120\"/>\n");
```

```
  // Write the note blocks in a for loop
  for ($index = 1; $index <= 8; $index++){
    fwrite($outfile,"      <note>\n");
    fwrite($outfile,"        <pitch>\n");
    fwrite($outfile,"          <step>$notestep[$index]</step>\n");
    if($notealter[$index] == "+1" || $notealter[$index] == "-1"){
      fwrite($outfile,"          <alter>$notealter[$index]</alter>\n");
    }
    fwrite($outfile,"          <octave>$noteoctave[$index]</octave>\n");
    fwrite($outfile,"        </pitch>\n");
    fwrite($outfile,"        <duration>1</duration>\n");
    fwrite($outfile,"        <type>eighth</type>\n");
    fwrite($outfile,"      </note>\n");
  }

  // write the rest of the measure
  fwrite($outfile, "    </measure>\n");
}

fwrite($outfile, "  </part>\n");
fwrite($outfile, "  <!--=========================================================--
>\n");
fwrite($outfile, "</score-partwise>\n");
fwrite($outfile, "\n");

fclose($outfile);
fclose($infile);

?>
```

Add The Melody To The Score

The output of melody2xml.php will be melody.musicxml or whatever you changed it to at the top of the program code. Import melody.musicxml into Finale and check it out. The file will import all on one line so choose Utilities, Fit Measures, 4, to make it readable.

Click to the left of the staff or hit Ctrl-a to highlight the entire melody. Then hit Ctrl-c to copy it to the clipboard.

Open the file sweet_mint_tea.musx that you created after generating the bass part. Click the first bar of the alto sax part to highlight it and then type Ctrl-v to paste the melody in.

To get a jazz swing sound select MIDI/Audio, Human Playback, Jazz. You will now be able to hear both the melody and bass parts.

Making It Better

Listening to the melodies generated by this algorithm there is an overwhelming feeling of sameness, an endless string of eighth notes. Although endless strings of eighth notes do happen in jazz and earlier styles of music most composers do use rests and notes with a longer duration like a quarter note or half note.

I will improve the algorithm that generates the melody with two simple changes.

Add Rests

Add a percentage rests parameter.

```
$percentagerests = 25;
```

Then the program picks a random number from 1 to 100 and if it's <= $percentagerests it gets an eighth rest instead of the next note picked by the current algorithm. This is implemented as an additional pass through the data leaving the existing code untouched.

Pseudo Code To Add Rests

```
Initialize percentagerests variable between 0 and 100.

Open a melody.txt file for input. (melody1.txt, melody2.txt, or whatever you called
your last melody.txt file)

Open a melody-rests.txt file for output. (melody1-rests.txt for instance)

Read the first line from the input file.

Generate a random number between 0 and 100.

If it's less than percentagerests variable
  Write "RST" to the output file.
else
  Write the first line to the output file.
endif

Close files.
```

Code To Add Rests

Working code, addrests.php, is included with the book's supplemental materials available at bac.kgpl.org. To run this program copy it into the same folder as your melody output file, in this case, melodyout.txt. Then open a terminal or command prompt and type this command.

```
php addrests.php
```

Then hit enter.

Sometimes the modifications are much easier to implement than the original algorithm. This is the easiest program yet and should be quite easy to follow and compare with the above pseudo code.

```
<?php

// Initialize percentage rests variable
$percentagerests = 25;

// input file
$inputfile = "melodyout.txt";

// open input file for reading
$infile = fopen($inputfile,"r");
```

```
// output file
$outputfile = "melody-rests25.txt";

// open output file for writing
$outfile = fopen($outputfile,"w");

// Read line from the input file
$line = fgets($infile);

while(! feof($infile)){
  // Compute random number
  $random = rand(1,99);

  // output a rest or the line just read
  if($random < $percentagerests){
    fwrite($outfile, "RST\n");
  }else{
    fwrite($outfile, $line);
  }

  // Read line from the input file
  $line = fgets($infile);
}

// Close files
fclose($infile);
fclose($outfile);

?>
```

After you run this program you will have a file called melody-rests25.txt that needs to be converted to musicXML so we can import it into Finale.

Generating MusicXML File With Rests

I used the above code to output melody-rests25.txt. The same as melodyout.txt but with rests added randomly throughout the file. The melody2xml.php does not take rests into account so it had to be modified. I call the modified file melody2xml-rests.php.

First I reverse engineered a Finale musx file with eighth notes and eighth rests to get the code for the note block when there is a rest. This text looks like this.

```
<note default-x="168">
  <rest/>
  <duration>1</duration>
  <voice>1</voice>
  <type>eighth</type>
</note>
```

I used the utility program, xml2php.php to turn this text into php write statements. (See **Using Code To Write Code** on page 36.) Here's the block of code including the conditional for the first note in the measure. I also removed the extra text in the note line

```
if($notestep[$index] == "R"){
  fwrite($outfile, "<note>\n");
  fwrite($outfile, "  <rest/>\n");
  fwrite($outfile, "  <duration>1</duration>\n");
  fwrite($outfile, "  <voice>1</voice>\n");
  fwrite($outfile, "  <type>eighth</type>\n");
  fwrite($outfile, "</note>\n");
}else{
```

I added this code into the for loop that writes the notes. I also fixed my indentation and added a closing bracket, }, to end the conditional after I write the note block. So I test for rests first in the for loop and then if it's not a rest I run the existing code to write the note.

Now I can output generated melodies with eighth note rests to musicXML.

I changed the output file name to melody-rests25.musicxml.

After you run melody2xml-rests.php you will have melody2xml-rests.musicxml. Import that into Finale and copy it into the score, sweet_mint_tea.musx.

The new program, melody2xml-rests, essentially replaces the first program, melody2xml.php, because it will work fine on files without rests too. Programmers have to decide whether to merge new features into an existing program or write another program instead.

Add Ties

Add a percentage ties parameter.
`$percentageties = 50;`

Anytime a rest is encountered the program picks a random number between 1 to 100 and if it's less than $percentageties the rest is replaced by the preceding note and a tie is added tying the two notes together creating a quarter note.

Since it would be possible to have two or more rests in a row it would also be possible to have tied notes longer than a quarter note.

The composer can change these parameters to find a rhythmic texture that is not too sparse and not too thick. Just like Goldilocks.

Pseudo Code For Adding Ties

Adding ties is a little trickier than adding rests because when you need to replace a rest with the previous note and add a tie to the previous note you have to not only change the current note but also read and write the previous note. Here is a first pass pseudo code that doesn't take implementation into account. It's just a description of the algorithm.

Loop through the notes skipping the first note.
  If the note is a rest generate a random number and compare it to the percentage ties variable.

  If the random number is less than the percentage ties variable read the previous note, change it to be tied, and set the current note to match the previous note.
End loop.

In principle this is simple enough. Because the program has to read and change the previous note we will put all the notes into an array. Now we will write pseudo code that deals with how this can actually work using an array to store and change the notes.

```
Initialize note array.

Open the input file for reading.

Read the input file into the note array, one note per array entry so we will have
note[0] -> A4, note[1] ->Bb4, note[2] -> RST, ... Count the number of notes while
reading the file into a total notes variable.

Close the input file.

Set an index counter to 1 skipping the first note.

While index counter is less than total notes
  Check if note array[index counter] is a rest.

  If it is a rest
    // Set current note to previous note
    Set note array[index counter] equal to note array[index counter - 1]

    // Set the previous note to have a tie (~ = tied)
    Add a ~ to the end of note array[index counter - 1]
  End if.

  Increment index counter.
End while

Open the output file for writing.
Loop through the array writing the notes into the output file, one note per line.
Close the output file.
```

Of course, the export to musicXML program will have to be changed again because we have not had to write ties until now.

Code For Adding Ties

Working code, addties.php, is included with the book's supplemental materials available at bac.kgpl.org. To run this program copy it into the same folder as your melody-rests output file, in this case, melody-rests25.txt. Then open a terminal or command prompt and type this command.

```
php addties.php
```

Then hit enter.

```
<?php

// Initialize percentage ties variable
$percentageties = 50;

// Enter input file name
$inputfile = "melody-rests25.txt";

// Open input file for reading
```

```
$infile = fopen($inputfile,"r");

// Enter output file name
$outputfile = "melody-rests25-ties50.txt";

// open output file for writing
$outfile = fopen($outputfile,"w");

// Read line from the input file and trim it
$line = trim(fgets($infile));

// Set counter to 0
$count = 0;

// Read input file into note array
while(! feof($infile)){
  // Write line into note array
  $notearray[$count] = $line;

  // Read line from the input file and trim it
  $line = trim(fgets($infile));

  // Increment counter
  $count++;
}

// Loop through the array changing current note and previous note when $random <
$percentageties
// Set $index to 1 instead of 0 to skip first note
$index = 1;

while($index < $count){
  // Check if current note is a rest
  if($notearray[$index] == "RST"){
    // Compute random number
    $random = rand(1,99);
    // Check if $random < $percentageties and change current and previous notes if
it is
    if($random < $percentageties){
      // Set current note to previous
      $notearray[$index] = $notearray[$index-1];

      // Add tie indicator (~) to previous note if it's not a rest
      if($notearray[$index-1] <> "RST"){
        $notearray[$index-1] .= "~";
      }
    }
  }
  // Increment index
  $index++;
}

// Write array to output file
$index = 0;
while($index < $count){
  //Write the output line with the \n newline added
  fwrite($outfile, $notearray[$index]."\n");
```

```
  // increment $index
  $index++;
}

// Close files
fclose($infile);
fclose($outfile);

?>
```

Naming Convention

When a file has rests I have added -restsNN to the filename where NN is the percentage of rests. When a file has rests and ties I have added -restsNN-tiesMM where NN is the percentage of rests and MM is the percentage of ties. This helps me to quickly evaluate the rhythmic texture against the parameters by putting these percentages right in the file name.

Generating MusicXML File With Rests And Ties

By reverse engineering the musicXML for ties we see that both the note tied from and the note tied to receive notation blocks. The first note gets a <tied type="start"/> and the second gets <tied type="stop"/>. So when we write out the musicXML and we encounter a tie we'll have to change both notes.

This musicXML text was exported from Finale for two eight notes tied together.

```
<note default-x="414">
  <pitch>
    <step>B</step>
    <octave>4</octave>
  </pitch>
  <duration>1</duration>
  <tie type="start"/>
  <voice>1</voice>
  <type>eighth</type>
  <stem default-y="-55">down</stem>
  <beam number="1">end</beam>
  <notations>
    <tied type="start"/>
  </notations>
</note>
<note default-x="511">
  <pitch>
    <step>B</step>
    <octave>4</octave>
  </pitch>
  <duration>1</duration>
  <tie type="stop"/>
  <voice>1</voice>
  <type>eighth</type>
  <stem default-y="-55">down</stem>
  <beam number="1">begin</beam>
  <notations>
    <tied type="stop"/>
  </notations>
</note>
```

You can see the tied type blocks, <tied type="start"/> and <tied type="stop"/>.

Pseudo Code For Melody With Rests And Ties

Starting with melody2xml-rests.php we will modify it so that it also outputs musicXML for tied eighth notes.

Instead of writing detailed pseudo code for a program we have already written twice I will explain the changes with code snippets in the Code text, immediately below.

Code For melody2xml-rests-ties.php

Working code, melody2xml-rests-ties.php, is included with the book's supplemental materials available at bac.kgpl.org. To run this program copy it into the same folder as your melody_rests-ties output file, in this case, melody-rests25-ties50.txt. Then open a terminal or command prompt and type this command.

```
php melody2xml-rests-ties.php
```

Then hit enter.

There are four places where the code from melody2xml-rests.php has to be changed. Instead of listing the entire program here I will just discuss the three code snippets that effect the change.

First we have to initialize a flag. Just before the measures loop we set $tieflag to 0. When we encounter a tie we will set this variable to 1 which will tell us that we need a stop tie notation block included for that note. If the current note doesn't have a tie the flag should be reset to 0.

Here's the line of code that initializes $tieflag.

```
// Remember whether the last note was tied using a variable, $tieflag
$tieflag = 0;
```

Second we have to detect whether the note is tied and set a variable if it's tied. We do this before we write the musicXML when we are setting variables for all the notes in the next measure. Also since we read the octave from the last character of the note in previous versions of this program we will have to adjust that to read from the next to the last character when there's a tie.

Here's the code snippet for doing this.

```
if(substr($note[$index], -1) == "~"){
  $noteoctave[$index] = substr($note[$index], -2, 1);
  $notetie[$index] = "~";
}else{
  $noteoctave[$index] = substr($note[$index], -1);
  $notetie[$index] = "";
}
```

This snippet replaces the line

```
$noteoctave[$index] = substr($note1, -1);
```

Third we have to write the correct musicXML for the ties. This affects both note blocks, the start note for the tie and the stop note for the tie. This snippet writes the start tie notation block if the $note1tie variable is set to ~.

```
// Add notation block for tie if needed
if($notetie[$index] == "~"){
  fwrite($outfile,"        <notations>\n");
  fwrite($outfile,"          <tied type=\"start\"/>\n");
  fwrite($outfile,"        </notations>\n");
}elseif($tieflag == 1){
  fwrite($outfile,"        <notations>\n");
  fwrite($outfile,"          <tied type=\"stop\"/>\n");
  fwrite($outfile,"        </notations>\n");
}
```

Finally the fourth code snippet manages $tieflag, making sure it is set correctly for the next note.

```
// Set the tie flag
// This variable will be remembered for the next note to stop tie
if ($notetie[$index] == "~"){
  $tieflag = 1;
}else{
  $tieflag = 0;
}
```

So we have to use two variables to control the tie notation blocks. We need to know if the current note starts a tie and we need to know if the previous note started a tie so we can end the tie on this note.

For readability I have only included the code that had to be changed to make the ties work. The entire program is included in the book's supplemental materials available at bac.kgpl.org. To run this program copy it into the same folder as your melody-rests-ties output file, in this case, melody-rests25-ties50.txt. Then open a terminal or command prompt and type this command.

```
php melody2xml-rests-ties.php
```

Then hit enter.

After you run melody2xml-rests-ties.php you will have melody2xml-rests25-ties50.musicxml. Import that into finale and copy it into the score, sweet_mint_tea.musx. This is the one I like the best.

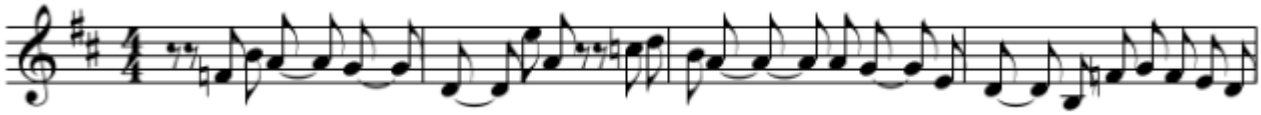
Improving The Data Structure

You can see that the data structure we are using for notes is becoming cumbersome. While C4 and Bb3 were fine now we have added C4~ and Bb3~ and we have also added a note RST which is another special case. If we carry this much farther a better data structure like associative arrays or an SQL database would be easier to deal with than a text file with coded character strings.

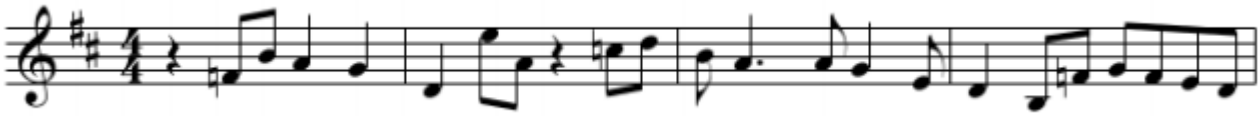
Readability

When we import the musicXML with rests and ties Finale will play the part correctly. If the part is printed and given to a musician it will be difficult to read. Longer notes like the quarter note are rarely shown as two eighth notes tied.

Also two eighth rests in sequence are almost always shown as a quarter note. Here's a sample from a musicXML import.



Here's the way musicians are used to seeing this.



Now it's possible to enhance the code to output musicXML that looks like this. This would also make the program much more difficult to write and understand. Since this is an example program used to clarify algorithmic composition this won't be covered in this book.

And it gets worse. Did you notice I said "rarely shown" and "almost always" in the paragraph above the parts? If the two eighth notes have a bar line in between they will be written as two eighth notes tied across the bar. Also if the two eighth notes lie on the upbeat of 2 and the downbeat of 3 in 4/4 time they are often written not as a quarter note but as two eighth notes tied to "expose" beat 3. The same uncertainty prevails in converting two eighth rests to a quarter rest. The truth is unlike musicXML, music notation was not defined by a standards committee but instead it grew and changed willy-nilly over the course of centuries in different countries and different cultures.

Rather than work on some difficult programming of limited utility I choose to manually update my parts in Finale to make them more readable. I can listen to them fine with non standard notation. When I prepare the parts to be read by a musician I give them a manual pass to make them easy for the musician to read. And I don't do this only when working with algorithmic composition and musicXML imports. I do this with every part of every piece I intend to have read. It makes no sense to use rehearsal time explaining notation to the musicians. Always do everything you can to make your parts easy to sight read and easy to learn.

Further Exploration

Experiment with different numbers for the percentage rests and percentage ties variables. The code is written so it is easy to change these variables and the names of the input and output files right at the top of the program text. If you choose high percentages for both rests and ties you will end up with many longer notes and longer rests. Think about changing these variables on the fly while the program is running.

Besides changing the algorithm it can also be very effective to change the data. I made another notes data set. The song I chose was "Mambo Bounce" from the Sonny Rollins Omnibook. I can hear a clear difference in the note choices between this data set and the first data set based on a Charlie Parker tune. I hear fewer non-diatonic notes and more repeated notes. I used the same percentages of rests and ties to make for a more direct comparison. It would be interesting to explore different percentages to see if you can tailor the texture of the melody to the note choices made with each data set. I have included these files in the supplemental materials.

```
sweet_mint_tea2.pdf
sweet_mint_tea2.musx
sweet_mint_tea2-garritan.mp3
sweet_mint_tea2-garritan.wav
```

These are the same filenames used for the first data set with the 2 added.

Try other data sets from other musicians and other genres. Try pulling melodies from classical music. J.S. Bach comes to mind as music that could make an interesting data set. Try writing your own melodies and using them as the input for this algorithm.

The example melody has no input from "Sweet Mint Tea" the song used to derive the bass part. This still works because they are in the same key. Try using the chord change from "Sweet Mint Tea" to influence the random selections by giving extra weight in the random choice to notes used in the bass line or to the four notes in the seventh chord from the change following each bar. This way the gravity of the chord pulls the melody towards the chords tonality.

Try deriving a different melody algorithm that uses only the songs chord change as input without using a database. Is it possible to write a deterministic algorithm to generate melodies like we did with the bass part?

Write algorithms that are not so reliant on just quarter notes and eighth notes. Quarter note triplets, eighth note triplets, and sixteenth notes are frequently used and can provide rhythmic variety to the generated tunes.

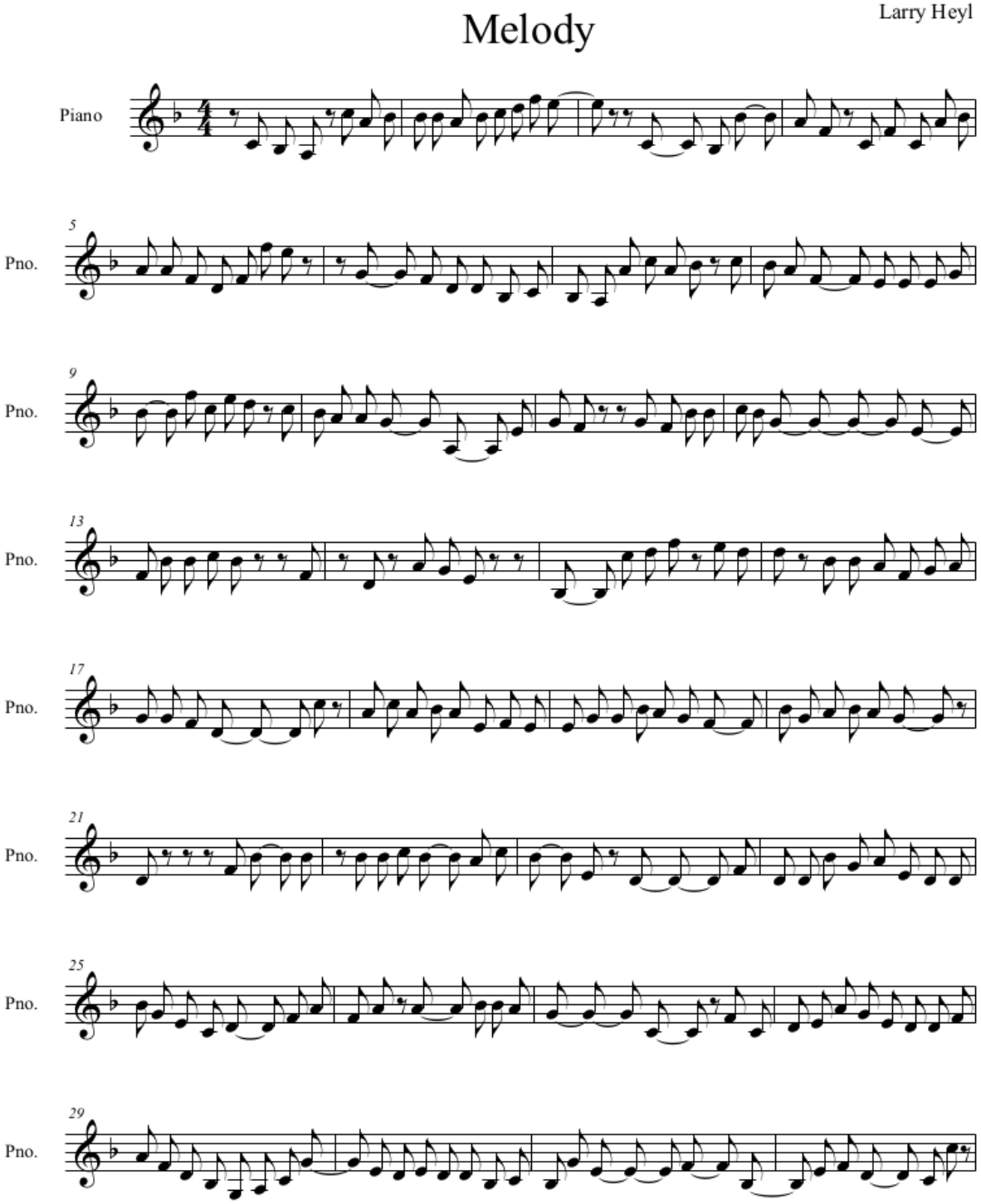
Use other scales and other modes. Do not limit yourself to writing algorithms that generate music that sounds like popular genres. Don't be afraid of experimental music and entirely new sounds. One of the powerful things about algorithmic composition is that composers can listen to even quite unusual generated pieces and become accustomed to them before they tweak and improve, making it better. Trust yourself and make your own decisions.

There's More Than Just Notes

These example programs have only dealt with notes. The bass lines are broken down into 4 quarter notes. The melody is generated from three note sequences. The rhythm parts, coming up next are expressed only as hits or rests. As long as the algorithmic composer only controls notes the results will not be musical.

Dynamics are marked on a score as mp or ff but musicians play varying dynamics in every phrase and it is possible for the algorithmic composer to actually specify the volume in detail that would be impossible to write on sheet music.
Even a single note is thought of as having attack, sustain, and decay. This note envelope can be selected by the algorithmic composer. They are just curves on the cartesian plane and algebra is great at drawing curves.

Tone and timbre may be more difficult to define but look at it this way. If it can be played on a synthesizer it can be quantified and described by algorithms.



Melody with rests and ties imported from musicXML file.

Rhythm

The idea behind the rhythm algorithm is to generate extended sections of interesting rhythms without repeating or sounding monotonous. In order to do this I use prime number theory.

I am going to restrict myself to discussing the counting numbers or positive integers. 1, 2, 3, 4, 5, ... where the three dots or ellipses can be read as "and so on." That is all we need for the algorithm since we will be thinking of the first note, the second note, the third note, and so on.

So we start with the idea of factors. A factor is just a multiplier in a multiplication problem. If we write 3*5 = 15 three and five are said to be factors of 15. Both 3 and 5 divide evenly into 15 and they can be seen as multipliers in a multiplication problem with the answer 15.

The other positive factors of 15 are 1 and 15. All numbers have as factors themselves and 1. If a number has only these two factors it is called prime. If a number has more than two factors it is not prime and it's called composite.

Fifteen is not prime because it has four factors, 1, 3, 5, and 15. Fifteen is composite.

Three is prime because three has only two factors, 1 and 3. Five is prime because five has only two factors, 1 and 5.

Since 3*5=15 and both 3 and 5 are prime we call this a prime factorization.

For every positive number there is exactly one prime factorization. This is a very important math theorem called the Fundamental Theorem of Arithmetic that has been proved. We are not going to deal with any math proofs in this book. If you are interested in how to prove this theorem there is a link in the glossary.

For a musician 8 is an interesting number because there are 8 eighth notes in a bar of 4/4 time. The prime factorization of eight is 2*2*2=8. Prime factorizations like this that only have one prime factor multiplied by itself are not useful for this algorithm.

There are many percussion patterns that are 1 bar long but there are very few pieces that repeat that pattern for the entire song. Most percussionists learn a beat, whether it is 1 bar long or longer, and then improvise on that beat. This improvisation provides variety and becomes part of the communication between the musicians and the audience. In orchestra pieces the composer usually writes out every percussion note and there is rarely improvisation in the parts. The composers provide variety in the parts they write.

This algorithm doesn't improvise. To provide variety this algorithm uses prime factors. Essentially I am using prime factors in the algorithm to add variety to the rhythm just like any composer does when writing a score.

To keep things simple we are going to generate one 32 bar part for the piece. I will explain the algorithm in terms of this 32 bar piece. The algorithm can generate pieces in any form and of any length.

We are going to write eighth note rhythms commonly counted "1 and 2 and 3 and 4 and". The drums hits on 1, 2, 3, and 4 are called down beats. The other four beats are called upbeats. The 2 and the 4 are called back beats. Clapping on 1 and 3 is

square. Clapping on 2 and 4 is hip. And the upbeats take the rhythmic interest to a whole different level. It helps to think about these things when laying out the math for the algorithm.

32 bars times 8 eighth notes per bar equals 256 eighth notes. The prime factorization of 256 is 2*2*2*2*2*2*2*2=256, commonly written 2^8 = 256 because looking at that row of twos it's hard to be certain just how many twos there are.

This is not suitable for this algorithm which requires a more complex prime factorization. I would like to have at least three distinct prime factors.

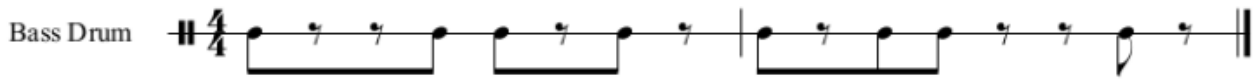
30 bars times 8 eighth notes equals 240 eighth notes. The prime factors of 240 are 2*2*2*2*3*5. This is much more interesting. I will use the algorithm to generate the first 30 bars of the 32 bar form. Then I will repeat the last two bars creating a rhythmic cadence or ending to the part.

In the first 30 bars every bar will be unique. Only the last two bars repeat the two bars before. We'll see if this is at all noticeable and if it feels like the part is ending which is the purpose of a rhythmic cadence.

The algorithm works like this. I will generate three rhythms for three instruments, kick drum, snare drum, and cymbal. One rhythm will be 16 eighth notes long. One will be three eighth notes long. And the third will be 5 eighth notes long.

So for the kick drum I devise a part 16 eighth notes that's 2 bars long like this.

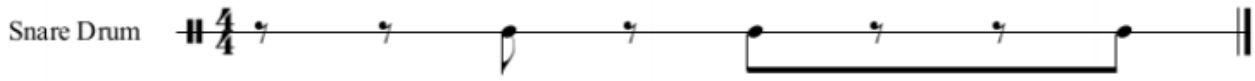
|x--xx-x-|x-xx--x-|



Since the bass drum part is 16 eighth notes long it fits perfectly in two bars and repeats every two bars.

For the snare drum I write a part 5 eighth notes long.

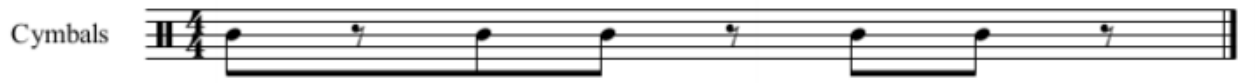
--x-x



The snare part is 5 eighth notes long. We see the 5 note part and then when it repeats we see only 3 of the next five notes in this bar. The next two notes begin the second bar and then the 5 notes repeat again. 5*8=40 so the five note rhythm crawls along for 40 eighth notes or 5 bars. Then the 5 bar pattern repeats.

And for the cymbal I write a part that is 3 eighth notes long.

x-x



Here we see the 3 note pattern repeat twice and the third repeat has only two notes

in this bar. 3*8=24 so after 24 eighth notes or 3 bars it comes out even and then those three bars repeat again.

All of these parts repeat until we hit bar 240. In those 240 bars every bar has a unique rhythm.

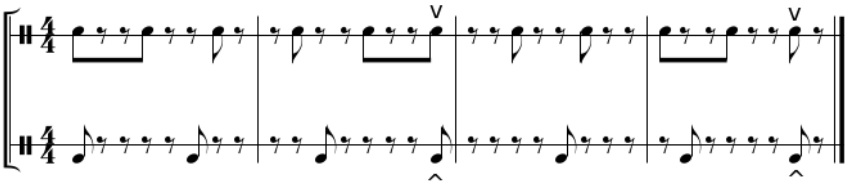
And how do we know that? Again, I am not going to offer a mathematical proof. Instead I am going to illustrate what is happening with a diagram.

Let's look at a 3 beat sequence and a 5 beat sequence.

x-- and x----

3*5*8 = 120 eighth notes. 120/8 = 15 bars. So the claim is that when these patterns are repeated through bar 15 every bar will have a unique rhythm. Here's the diagram and music notation.

x--x--x--x--x--
x----x----x----



I have marked the repeat points on this sheet music. Notice how each repetition of 15 beats crawls back 1 beat every two bars. After 15 bars the repetition again begins on beat one of bar 16 and the whole pattern repeats creating a 15 bar pattern.

At note 1 of bar 16, both drums hit on the one. It is easy to see that, except for the first note, at no other point in the 15 bars are both drums being hit on beat 1. The entire pattern repeats every 15 bars and it does not repeat until the 15 bars are complete. Then the whole pattern repeats at bar 16.

The bass drum is written in a 16 note pattern or two bars. The snare and the cymbal are written in a 15 note pattern, one note shy of two bars. The snare and the cymbal start on note one and repeat every 15 notes ending on note 15, note 30, note 45 ... Let me write the end notes of the repeats.

15-30-45-60-75-90-105-120-135-150-165-180-198-210-225-240

Not coincidentally the two bar pattern for the bass part also ends at note 240. The ending notes are.

16-32-48-64-80-96-112-128-144-160-176-192-208-224-240

We are counting eighth notes. You can see that these repeats all end on a different eighth note beat until beat 240 is reached.

As mentioned above 240 eighth notes is 30 bars because 240/8 = 30. These three simple patterns were chosen in lengths of 3, 5, and 16 because 3*5*16 = 240 generating 30 bars of rhythm with each bar unique. Then repeating the last two bars we get the desired length of 32 bars to fit the form of "Sweet Mint Tea".

We will generate each part 30 bars long and then we will manually repeat the last two bars in Finale. When combined with the bass and melody generated in the last two sections we will have a 32 bar piece with 5 instruments, bass, sax, kick drum, snare drum, and cymbal.

Parts Overview

The walking bass part is generated using a simple rule set that generates a bass line for the chords used in "Sweet Mint Tea".

The melody part is generated using random selection from a database of three note sequences. This melody has no connection to "Sweet Mint Tea" except that it is also in the key of F. In Further Explorations in the melody section it is suggested that we might be able to make it better by allowing the chords to influence the random selection.

The three drum parts are also deterministic using prime number theory to insure unique rhythms in every measure. There are many other interesting math concepts that can be used in algorithmic composition.

So you can see the progression in algorithmic sophistication from one part to the next.

When we finish implementing the percussion algorithm we will have a complete piece. Also we will have improved each part with Making It Better and include suggestions for Further Explorations. You can use the code base developed in this book in your own algorithmic compositions perhaps not just making it better but actually devising your own algorithms from math and music theory.

But now we're going to implement our rhythm algorithm.

Pseudo Code For The Rhythm Algorithm

I will write and output to musicXML three separate parts. The same algorithm will work on all three parts. This same piece of code can generate no end of complexity, it will work with any part of any length 2 or greater repeated any number of times.

Here's the pseudo code.

```
Input the beat.
$beat = "--x-x";

Get the length of the beat.
$beatlength = strlen($beat);

Input the number of eighth notes.
$numnotes = 240;

Compute the number of repeats.
$numrepeats = $numnotes/$beatlength;

Initialize index.
$index = 0;

Initialize output string.
$outstring = "";

Loop and fill the output string.
While $index < $numrepeats
  Add the beat to the end of the output string.
  $outstring .= $beat;
```

```
  Increment the index.
  $index++;
End while loop.

Open output file, rhythmout.txt.
$outfile = fopen("rhythmout.txt","w");

Initialize the loop counter.
$counter = 0;

Loop and count eighth notes while writing the file.
While($counter < $numberofnotes)
  Output character to file.
  fwrite(outputfile,substr($outstring,$counter,1));

  Increment $counter.
  $counter++;
End while loop.

Close output file.
fclose(outputfile)
```

Code For The Rhythm Part

Working code is included with the book's supplemental materials available at bac.kgpl.org.

This is the easiest piece of code so far. Of course counting the notes out into bars will occur in the program that converts the output file into musicXML but there's nothing difficult about that either. This program has no input file. The input is all in the settings at the top of the program. To run this program copy it into a folder and change the settings if desired. Then open a terminal or command prompt and type this command.

```
php makerhythm.php
```

Then hit enter.

```
<?php

// Input the beat.
// Change this line to change the beat.
$beat = "x-x";

// Input the number of eighth notes.
// Change this line to change the length of the rhythm.
$numnotes = 240;

// Input the name of the output file
// Change this line to change the name of the output file.
$outfile = "cymbalx-x.txt";

// Open output file
$outfile = fopen($outfile,"w");

// Get the length of the beat.
$beatlength = strlen($beat);
```

```
// Compute the number of repeats.
$numrepeats = $numnotes/$beatlength;

// Initialize index.
$index = 0;

// Initialize output string.
$outstring = "";

// Loop and fill the output string.
while($index < $numrepeats){
  // Add the beat to the end of the output string.
  $outstring .= $beat;

  // Increment the index.
  $index++;
}

// Initialize the loop counter.
$counter = 0;

// Loop and count eighth notes while writing the file
while($counter < $numnotes){
  // Output character to file.
  fwrite($outfile,substr($outstring,$counter,1)."\n");

  // Increment $counter.
  $counter++;
}

// Close output file;
fclose($outfile);

?>
```

If unchanged the program will output a file called snare--x-x.txt. This is the rhythm file that will be the input for the program to convert it to musicXML.

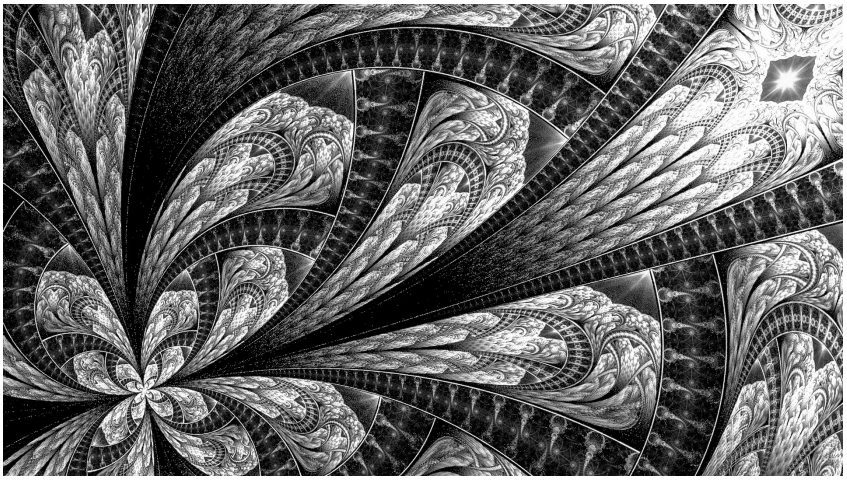
This algorithm is based in prime number theory, an important field in mathematics. It has no random element so it is deterministic meaning that the derived parts are exactly repeatable, the same inputs will always result in the same rhythms. Yet it provides for endless complexity in rhythm parts of any length.

It could also be used to sequence dynamics, articulation, tone, or any other parameter and provide regular, long term variations that only repeat after the number of notes the composer chooses. Like much of mathematics the algorithm displays endless complexity from a few simple rules.

In the addendum we'll look at fractal geometry and how fractals have been applied to algorithmic composition. Using fractal geometry it is possible to draw figures of infinite complexity applying rules that can be explained in two or three sentences. Group Theory and Ring Theory also provide amazing symmetries that could be used in algorithmic composition.

This fractal image by Barbara A Lane from Pixabay.

Infinite complexity from simple expressions.



This illustrates the power of using math in your algorithmic compositions.

Documenting Naming Conventions

It's important for the filenames to contain information about the contents of the files. Programmers use naming conventions so they can glance at a set of files and know what each file is and what it's purpose is. Because programmers forget stuff from time to time they document naming conventions. Here is the document for the naming conventions and the functionality of the files used generating the rhythm parts.

The standard prefix for the set of files that are used to generate one part will be instrumentrhythm. For example.

kickx--xx-x-x-xx--x- or snare--x-x

This groups the instruments together in an alphabetical directory listing and it includes the important information about this set of files, that it is generating this rhythm for this instrument.

File extensions are used to indicate the type of file. In generating a rhythm part we use files with these extensions.

.txt - text file
.musx or .mus - Finale file
.php - program file
.musicxml or .xml - a musicXML file

Here is the set of files used generating the snare part.

snare--x-x.txt
snare-_template.musx
snare_template.musicxml
snare_template.php
snare--x-x.musicxml
Here is how each file is used and where it occurs in the work flow.

snare--x-x.txt

This is the data file for the part. It is created by makerhythm.php

snare_template.musx

This is a Finale file that is used to reverse engineer the xml template. It is 2 bars long and contains rhythms similar to rhythms in the part defined in snare--x-x.txt. It is created for the instrument and includes composer, copyright, and any other meta data desired. The purpose of this file is to output the template for that instrument and the meta data.

snare_template.musicxml

This is the musicXML template file exported from the Finale file. This template is the input to the xml2php.php program that turns each line into a write statement in php. This procedure is described in the **Using Code To Write Code** section of this book on page 36.

snare_template.php

This program is created by running xml2php.php with snare_template.musicxml as the input file. It's the starting point for reverse engineering the musicXML output from Finale and is where the musicXML write statements come from for rhythm2xml.php.

rhythm2xml.php

This is the program made when snare_template.php is modified to read snare--x-x.txt. It outputs snare--x-x.musicxml, the musicXML file for the completed part.

This file can be imported into Finale and then printed or played. The modifications involve changing the header code, reading the input file, and putting the measure block output into a loop. This process is detailed in the pseudo code and code examples. The program is similar to melody2xml.php.

snare--x-x.musicxml

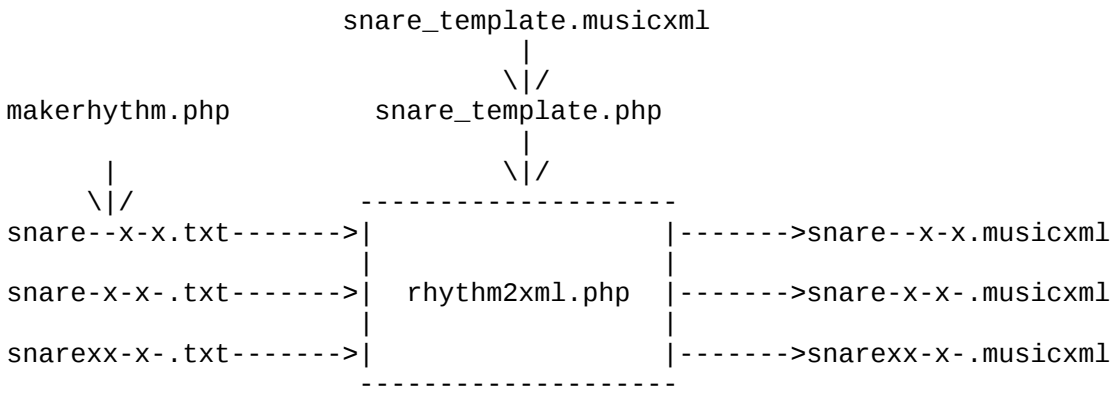
This output from rhythm2xml.php is the completed part ready to be imported and used.

Here is the code snippet from xml2php.php modified for these filenames.

```
// Change input and output files here.
// This is the template file to be used for reverse engineering the xml.
$inputfile = "snare_template.musicxml";
// This is the name of the program that is the output of this program.
$outputphp = "snare_template.php";
// This filename is used in the program that outputs the xml.
$outputfile = "snare_template-out.musicxml"; // use the xml or the musicxml
extension
```

If only the rhythm is changed the template parts above can be skipped. A newly generated text file like snare-x-x-.txt can be set as the input to the snare_template.php file and the output can be changed to snare-x-x-.musicxml without redoing the template or rewriting the snareout.php program.

This naming convention is used to organize and describe these files. This document describes the naming convention and the workflow that produces the finished musicXML file.



Above is an ASCII flowchart of the workflow using the naming conventions. snare_template.musicxml, snare_template.php, and rhythm2xml.php only need to be made once. The .txt files are generated from the makerhythm.php program. Then rhythm2xml.php takes the .txt files as input and generates the .musicxml files.

Converting The Rhythm Part To MusicXML

The pseudo code for converting the rhythm part to musicXML is the same as for the melody part. See the chapter **Pseudo code For Melody Algorithm** on page 47.

Programming computers can be difficult and time consuming. On the other hand if you are writing a program to do something similar to what you have done before much of your code will be reusable saving work and effort. Modifying existing code is easier and quicker than writing code from scratch. Please feel free to use any of the code that comes with this book in your projects.

Code For Converting The Rhythm Part To MusicXML

Working code, rhythm2xml.php, is included with the book's supplemental materials available at bac.kgpl.org. To run this program copy it to a folder containing a rhythm data file like snare--x-x.txt. Change the parameters at the top, if necessary. Then open a terminal or command prompt and type this command.

```
php rhythm2xml.php
```

Then hit enter.

```
<?php
// Set the number of measures.
$numberofmeasures = 30;

// Set the input file.
$inputfile = "snare--x-x.txt";

// Open the input file.
$infile = fopen($inputfile,"r");

// Set the output file.
$outputfile = "snare--x-x.musicxml";

// Open the output file.
$outfile = fopen($outputfile,"w");
```

```
// Write the header lines.
fwrite($outfile, "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?>\n");
fwrite($outfile, "<!DOCTYPE score-partwise PUBLIC \"-//Recordare//DTD MusicXML 3.1
Partwise//EN\" \"http://www.musicxml.org/dtds/partwise.dtd\">\n");
fwrite($outfile, "<score-partwise version=\"3.1\">\n");
fwrite($outfile, "  <movement-title>snare_template</movement-title>\n");
fwrite($outfile, "  <identification>\n");
fwrite($outfile, "    <creator type=\"composer\">Larry Heyl</creator>\n");
fwrite($outfile, "    <rights>© by Larry Heyl, 2021</rights>\n");
fwrite($outfile, "    <encoding>\n");
fwrite($outfile, "      <software>Finale v26.3 for Windows</software>\n");
fwrite($outfile, "      <encoding-date>2021-04-03</encoding-date>\n");
fwrite($outfile, "      <supports attribute=\"new-system\" element=\"print\"
type=\"yes\" value=\"yes\"/>\n");
fwrite($outfile, "      <supports attribute=\"new-page\" element=\"print\"
type=\"yes\" value=\"yes\"/>\n");
fwrite($outfile, "      <supports element=\"accidental\" type=\"yes\"/>\n");
fwrite($outfile, "      <supports element=\"beam\" type=\"yes\"/>\n");
fwrite($outfile, "      <supports element=\"stem\" type=\"yes\"/>\n");
fwrite($outfile, "    </encoding>\n");
fwrite($outfile, "  </identification>\n");
fwrite($outfile, "  <defaults>\n");
fwrite($outfile, "    <scaling>\n");
fwrite($outfile, "      <millimeters>7.2319</millimeters>\n");
fwrite($outfile, "      <tenths>40</tenths>\n");
fwrite($outfile, "    </scaling>\n");
fwrite($outfile, "    <page-layout>\n");
fwrite($outfile, "      <page-height>1545</page-height>\n");
fwrite($outfile, "      <page-width>1194</page-width>\n");
fwrite($outfile, "      <page-margins type=\"both\">\n");
fwrite($outfile, "        <left-margin>140</left-margin>\n");
fwrite($outfile, "        <right-margin>70</right-margin>\n");
fwrite($outfile, "        <top-margin>70</top-margin>\n");
fwrite($outfile, "        <bottom-margin>70</bottom-margin>\n");
fwrite($outfile, "      </page-margins>\n");
fwrite($outfile, "    </page-layout>\n");
fwrite($outfile, "    <system-layout>\n");
fwrite($outfile, "      <system-margins>\n");
fwrite($outfile, "        <left-margin>0</left-margin>\n");
fwrite($outfile, "        <right-margin>0</right-margin>\n");
fwrite($outfile, "      </system-margins>\n");
fwrite($outfile, "      <system-distance>121</system-distance>\n");
fwrite($outfile, "      <top-system-distance>70</top-system-distance>\n");
fwrite($outfile, "    </system-layout>\n");
fwrite($outfile, "    <appearance>\n");
fwrite($outfile, "      <line-width type=\"stem\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"beam\">5</line-width>\n");
fwrite($outfile, "      <line-width type=\"staff\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"light barline\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"heavy barline\">5</line-width>\n");
fwrite($outfile, "      <line-width type=\"leger\">1.0807</line-width>\n");
fwrite($outfile, "      <line-width type=\"ending\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"wedge\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"enclosure\">0.918</line-width>\n");
fwrite($outfile, "      <line-width type=\"tuplet bracket\">0.918</line-width>\n");
fwrite($outfile, "      <note-size type=\"grace\">60</note-size>\n");
fwrite($outfile, "      <note-size type=\"cue\">60</note-size>\n");
```

```
fwrite($outfile, "      <distance type=\"hyphen\">120</distance>\n");
fwrite($outfile, "      <distance type=\"beam\">7.5</distance>\n");
fwrite($outfile, "      <glyph
type=\"percussion-clef\">unpitchedPercussionClef1</glyph>\n");
fwrite($outfile, "    </appearance>\n");
fwrite($outfile, "    <music-font font-family=\"Maestro,engraved\" font-
size=\"20.5\"/>\n");
fwrite($outfile, "    <word-font font-family=\"Times New Roman\" font-
size=\"10.25\"/>\n");
fwrite($outfile, "  </defaults>\n");
fwrite($outfile, "  <credit page=\"1\">\n");
fwrite($outfile, "    <credit-type>title</credit-type>\n");
fwrite($outfile, "    <credit-words default-x=\"632\" default-y=\"1475\" font-
size=\"24\" justify=\"center\" valign=\"top\">snare_template</credit-words>\n");
fwrite($outfile, "  </credit>\n");
fwrite($outfile, "  <credit page=\"1\">\n");
fwrite($outfile, "    <credit-type>composer</credit-type>\n");
fwrite($outfile, "    <credit-words default-x=\"1122\" default-y=\"1407\" font-
size=\"12\" justify=\"right\" valign=\"top\">Larry Heyl</credit-words>\n");
fwrite($outfile, "  </credit>\n");
fwrite($outfile, "  <credit page=\"1\">\n");
fwrite($outfile, "    <credit-type>rights</credit-type>\n");
fwrite($outfile, "    <credit-words default-x=\"632\" default-y=\"53\" font-
size=\"10\" justify=\"center\" valign=\"bottom\">© by Larry Heyl, 2021</credit-
words>\n");
fwrite($outfile, "  </credit>\n");
fwrite($outfile, "  <credit page=\"1\">\n");
fwrite($outfile, "    <credit-type>part name</credit-type>\n");
fwrite($outfile, "    <credit-words default-x=\"140\" default-y=\"1478\" font-
size=\"12\" valign=\"top\">Score</credit-words>\n");
fwrite($outfile, "  </credit>\n");
fwrite($outfile, "  <part-list>\n");
fwrite($outfile, "    <score-part id=\"P1\">\n");
fwrite($outfile, "      <part-name>Snare Drum</part-name>\n");
fwrite($outfile, "      <part-abbreviation>S.Dr.</part-abbreviation>\n");
fwrite($outfile, "      <score-instrument id=\"P1-I1\">\n");
fwrite($outfile, "        <instrument-name>ARIA Player</instrument-name>\n");
fwrite($outfile, "        <instrument-sound>drum.snare-drum</instrument-sound>\n");
fwrite($outfile, "        <solo/>\n");
fwrite($outfile, "        <virtual-instrument/>\n");
fwrite($outfile, "      </score-instrument>\n");
fwrite($outfile, "      <score-instrument id=\"P1-X1\">\n");
fwrite($outfile, "        <instrument-name>Snare Drum</instrument-name>\n");
fwrite($outfile, "      </score-instrument>\n");
fwrite($outfile, "      <score-instrument id=\"P1-M71\">\n");
fwrite($outfile, "        <instrument-name>MIDI72</instrument-name>\n");
fwrite($outfile, "      </score-instrument>\n");
fwrite($outfile, "      <midi-device>ARIA Player</midi-device>\n");
fwrite($outfile, "      <midi-instrument id=\"P1-I1\">\n");
fwrite($outfile, "        <midi-channel>10</midi-channel>\n");
fwrite($outfile, "        <midi-program>1</midi-program>\n");
fwrite($outfile, "        <volume>80</volume>\n");
fwrite($outfile, "        <pan>0</pan>\n");
fwrite($outfile, "      </midi-instrument>\n");
fwrite($outfile, "      <midi-instrument id=\"P1-X1\">\n");
fwrite($outfile, "        <midi-channel>10</midi-channel>\n");
fwrite($outfile, "        <midi-program>1</midi-program>\n");
```

```
fwrite($outfile, "        <midi-unpitched>61</midi-unpitched>\n");
fwrite($outfile, "        <volume>80</volume>\n");
fwrite($outfile, "        <pan>0</pan>\n");
fwrite($outfile, "      </midi-instrument>\n");
fwrite($outfile, "      <midi-instrument id=\"P1-M71\">\n");
fwrite($outfile, "        <midi-channel>10</midi-channel>\n");
fwrite($outfile, "        <midi-program>1</midi-program>\n");
fwrite($outfile, "        <midi-unpitched>72</midi-unpitched>\n");
fwrite($outfile, "        <volume>80</volume>\n");
fwrite($outfile, "        <pan>0</pan>\n");
fwrite($outfile, "      </midi-instrument>\n");
fwrite($outfile, "    </score-part>\n");
fwrite($outfile, "  </part-list>\n");
fwrite($outfile, "  <!--=========================================================--
>\n");
fwrite($outfile, "  <part id=\"P1\">\n");

// Initialize measure number to 0. It will be incremented to 1 at the top of the
loop.
$measure = 0;

// Write the measures.
while($measure < $numberofmeasures ){
  $measure++;

  // Read eighth notes from the input file. - is rest x is hit
  for ($index = 1; $index <= 8; $index++){
    $note[$index] = trim(fgets($infile));
  }

  // Write the musicXML for this measure. Insert the measure number, $measure, in
the next line.
  fwrite($outfile, "    <measure number=\"$measure\" width=\"269\">\n");
  fwrite($outfile, "      <print>\n");
  fwrite($outfile, "        <system-layout>\n");
  fwrite($outfile, "          <system-margins>\n");
  fwrite($outfile, "            <left-margin>70</left-margin>\n");
  fwrite($outfile, "            <right-margin>0</right-margin>\n");
  fwrite($outfile, "          </system-margins>\n");
  fwrite($outfile, "          <top-system-distance>197</top-system-distance>\n");
  fwrite($outfile, "        </system-layout>\n");
  fwrite($outfile, "        <measure-numbering>system</measure-numbering>\n");
  fwrite($outfile, "      </print>\n");
  fwrite($outfile, "      <attributes>\n");
  fwrite($outfile, "        <divisions>2</divisions>\n");
  fwrite($outfile, "        <key>\n");
  fwrite($outfile, "          <fifths>-1</fifths>\n");
  fwrite($outfile, "          <mode>major</mode>\n");
  fwrite($outfile, "        </key>\n");
  fwrite($outfile, "        <time>\n");
  fwrite($outfile, "          <beats>4</beats>\n");
  fwrite($outfile, "          <beat-type>4</beat-type>\n");
  fwrite($outfile, "        </time>\n");
  fwrite($outfile, "        <instruments>2</instruments>\n");
  fwrite($outfile, "        <clef>\n");
  fwrite($outfile, "          <sign>percussion</sign>\n");
  fwrite($outfile, "        </clef>\n");
```

```
  fwrite($outfile, "        <staff-details>\n");
  fwrite($outfile, "          <staff-lines>1</staff-lines>\n");
  fwrite($outfile, "        </staff-details>\n");
  fwrite($outfile, "      </attributes>\n");
  fwrite($outfile, "      <sound tempo=\"120\"/>\n");

  // Write the note blocks for this measure.
  for ($index = 1; $index <= 8; $index++){
    if($note[$index] == "x"){
      fwrite($outfile, "      <note>\n");
      fwrite($outfile, "        <unpitched>\n");
      fwrite($outfile, "          <display-step>E</display-step>\n");
      fwrite($outfile, "          <display-octave>4</display-octave>\n");
      fwrite($outfile, "        </unpitched>\n");
      fwrite($outfile, "        <duration>1</duration>\n");
      fwrite($outfile, "        <instrument id=\"P1-X1\"/>\n");
      fwrite($outfile, "        <voice>1</voice>\n");
      fwrite($outfile, "        <type>eighth</type>\n");
      fwrite($outfile, "      </note>\n");
    }else{
      fwrite($outfile, "      <note>\n");
      fwrite($outfile, "        <rest/>\n");
      fwrite($outfile, "        <duration>1</duration>\n");
      fwrite($outfile, "        <voice>1</voice>\n");
      fwrite($outfile, "        <type>eighth</type>\n");
      fwrite($outfile, "      </note>\n");
    }
  }

  // Write the end of the measure.
  fwrite($outfile, "    </measure>\n");
}
fwrite($outfile, "  </part>\n");
fwrite($outfile, "  <!--=========================================================--
>\n");
fwrite($outfile, "</score-partwise>\n");
fwrite($outfile, "\n");

// Close the files.
fclose($infile);
fclose($outfile);

?>
```

Importing Percussion

Finale v26 imports the snare part fine. It says snare drum and sounds like a snare drum.

When I copy the snare part into the Sweet Mint Tea score the notes are below the staff and they sound like piano. I used the Solo button in the Mixer to play only snare drum without the other parts playing. The reason it plays piano is because of the default synthesizer Device, which on my system is Microsoft GS Wavetable. In the score manager I changed that to SmartMusic SoftSynth 2 and I changed the Sound to Standard. After I changed the midi settings it played a drum part but not a snare drum part. I selected the entire snare part by clicking to the left of the snare staff. Then from the menu I chose Utilities, Transpose Percussion Notes and

Finale opened a window with two columns, Note In Selected Region and Change To Note Type. I highlighted that line by clicking on it and this activated a drop down box next to Change Selected Note To. I clicked on the drop down box and selected in order, Snare Drums, Snare Drum, Snare Drum. Then the part started sounding like a snare drum and it was placed correctly just above the staff.

Although this seems like a lot of work to import a percussion part it is actually good news. It makes it easy to experiment with different percussion instruments just by using the Transpose Percussion Notes feature. And it also means that we can use the program rhythm2xml.php to convert the rhythms for any non-pitched percussion instrument since we are going to have to select the instrument in the score anyway.

So that is what I did with the kick drum and cymbal parts to complete the score. First I modify makerhythm.php for the beat and the output file. Then I use rhythm2xml.php to convert that part to musicXML, again changing the input and output files. After I import that musicXML file into Finale I copy and paste the part into the score. And then I change the instrument. I used the Garritan Fusion Drum Kit for the percussion parts.

You can also use the mixer to adjust volume levels and the pan control to place the instrument on the stereo image. I panned the sax left and the bass right leaving the drums in the middle.

I recorded both the midi render and the garritan render using the Export, Audio feature. Midi will output direct to mp3. The Garritan render only outputs to wav. I did some audio post production on both of these files normalizing the output and then exported the Garritan .wav file to .mp3. All three audio files are included in the supplemental materials, sweet_mint_tea-midi.mp3, sweet_mint_tea-garritan.wav, and sweet_mint_tea-garritan.mp3.

Making It Better

Changing The Data Instead Of The Algorithm

This algorithm can generate extremely long complex rhythms with more than three instruments. The composer can experiment with different patterns and different pattern lengths. By choosing pattern lengths that are relatively prime the composer can control how many beats or bars there will be before it repeats. By controlling the sparseness or density of the individual parts the composer controls the rhythmic texture.

Use Different Instruments

I like cowbell or agogo, wood blocks, and handclaps. It's easy to change instruments in Finale using the Transpose Percussion feature described above.

Additional Example Rhythms

The musicXML files generated by these example rhythms are included with the book's supplemental materials available at bac.kgpl.org.
This is where the naming conventions really come in handy.

Try these.

```
kickx--xx--xx-x-x--x   16 eighth notes
snare--x---x            7 eighth notes
cymbalx-xx-             5 eighth notes
```

5*7*16 = 560 eighth notes. 70 measures without repeating a rhythm so this would work for two choruses of "Sweet Mint Tea".

The prime factors do not have to be divisible by 2. If you use only primes that are not even numbers you still get the eight eighth notes in 4/4 time. If you do this it's helpful if the bass or some other part is emphasizing the one or the listener can get lost in the chaos.

```
kickx-x-x              5 eighth notes
snare--x--x-           7 eighth notes
cymbal-xx              3 eighth notes
```

3*5*7*8 = 840. 105 measures without repeating a rhythm.

Be sure to change the number of beats and number of bars variables as well as the beat and filename variables.

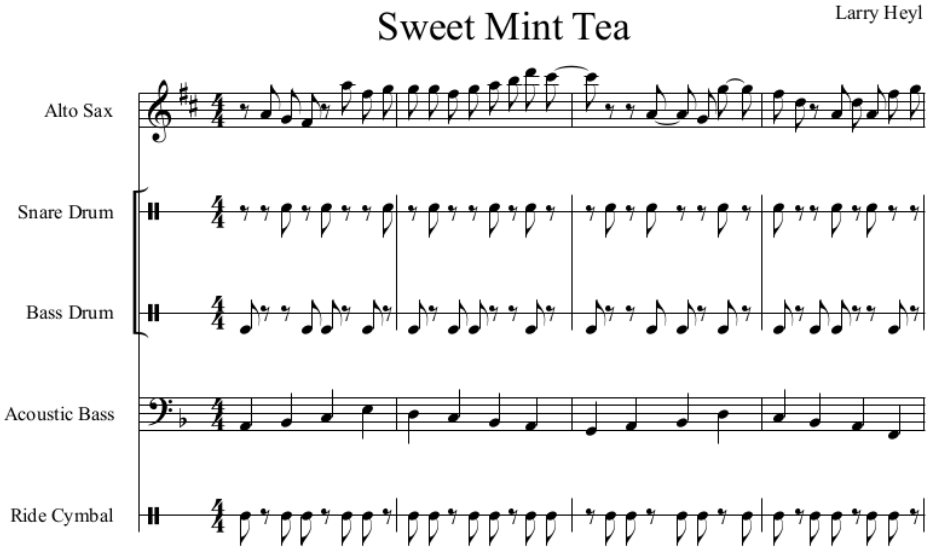
These examples are included as .wav and .mp3 files in the supplemental materials named polyrhythm1, polyrhythm2, and polyrhythm2alt.

Further Exploration

Try some other polyrythmic techniques like 3 against 2 or 7 against 5 where the two parts complete at the same time one part playing 7/8 and the other playing 5/8 but they always hit the same downbeat. So the parts are in different time signatures and at different tempos to compensate.

Writing an algorithm to play the above polyrhythms can be less work than rehearsing and playing the rhythms. This is true of many advanced percussion concepts. With a computer program it is possible for several instruments to constantly shift the tempo staying together or each part taking their own tempo.

Instead of scoring individual lines for the kick, snare, and cymbal it is possible to score all three instruments and more on a drum set staff. Not all drum set players read drum set staff well but some do. Generating readable drum set parts when converting to musicXML would be much easier to read than the separate drum lines generated by these example programs.



The entire score, sweet_mint_tea.musx, is included in the supplemental materials.

Mix It Up

Not everyone is a programmer. And not all algorithmic composition requires programming. At this point there's quite a bit of algorithmic composition we can do without writing any more code.

We can derive two bass lines, one with repeated notes and one with no repeated notes. We can change the input file to another song as long as the chords are all church modes and there is one chord per bar. We can change the length of the piece. I chose 32 bars for simplicity. Most compositions are longer.

We can derive melodies that are an endless string of eighth notes. We can add rests. We can add rests and ties. We can pull from two data sets for interval values. It would be easy to add more data sets from other pieces of music or devised by the composer. Every time we make a melody it's different. We can generate many different melodies and then select and add to the score the parts we like most placing them where we want.

We can generate rhythms of any length that never repeat. Or we can generate several shorter rhythms and then place them in the score to create a rhythm section that varies but also contains repetition. We can assign different instruments to the rhythm parts until we get just the texture we are looking for.

We can print the sheet music and rehearse ensembles. We can add dynamics and articulation to the sheet music manually. We can tell the musicians exactly how we want the parts played and help them achieve the desired blend.

All without writing another line of code and all with just the three algorithmic techniques used as examples in this book.

So I hope that programmers will extend these programs and I have them licensed with the GPL open source license so that they are legal for reuse. Even more, I hope that programmers will take inspiration from these simple programs and devise their own algorithms and write their own code.

Three Pieces

Here are some links to three of my algorithmic compositions. Some of these used computer programming and on some of them I worked the algorithm by hand. This is always an option for composers who are not programmers. Describe your algorithm. Write pseudo code for the manual process. Follow the pseudo code by hand doing necessary computations on paper and throwing percentile dice to determine probabilities.

These pieces were played by student ensembles at recitals or at a reading. "Waterfalls" was rehearsed, read, and recorded during a one hour band rehearsal. Thanks to Dr. Daniel Tacke, Dr. Ken Carroll, Dr. Derek Jenkins, and Mr. Craig Collison for helping with these recordings.

More details about the performances including personnel are listed on the web pages.

"Sympatico" recorded at the ASU Fine Arts Recital Hall on October 28, 2014.
https://archive.org/details/hl2014-10-28

"Even Dozen" recorded at Arkansas State University on November 13, 2015.
https://archive.org/details/hl2015-11-13

"Waterfalls" recorded at the ASU Band Room on April 19, 2019.
https://archive.org/details/hl2019-04-19

Thanks to the Live Music Archive at archive.org for hosting these pieces.

Is It Jazz?

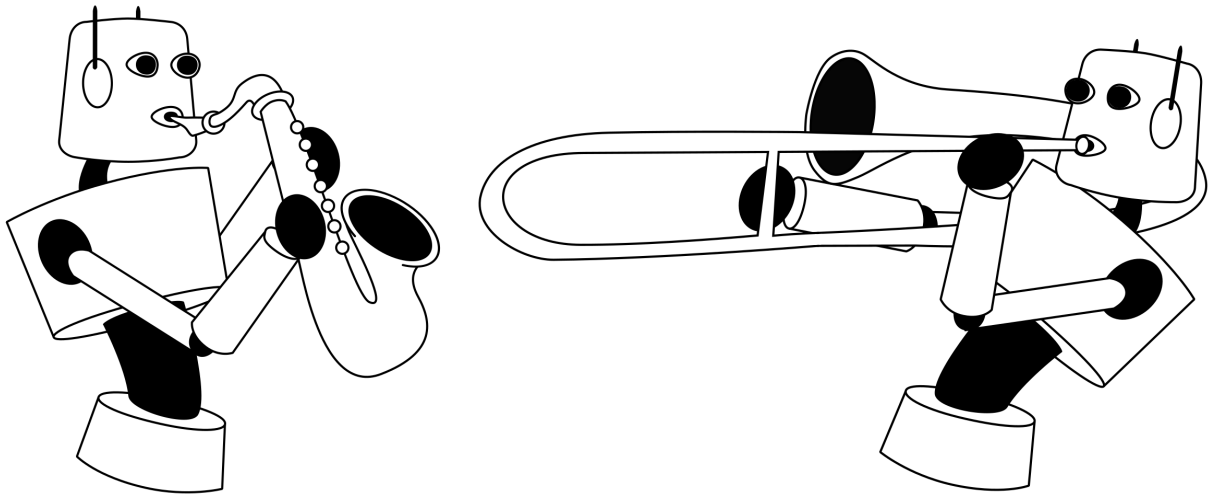
Ok, "Sweet Mint Tea" sounds a little bit like jazz. And clearly that's intentional. The walking bass and the exotic melody line sounding like something you heard before but not exactly the same as anything you ever heard. And on the drums, the shimmering cymbals, the upbeat snare hits, and the actual kick drum part holding down a two bar pattern and gluing everything together. It works.

But it's not jazz. Jazz is an improvisational art form where even when a horn is taking a solo the entire group is improvising supporting the solo and pushing the band forward. Jazz musicians communicate among themselves and with the audience. Jazz is live!

I am not saying that algorithmic composers will never write jazz. Computer programs win in games of Chess and Go against Grandmasters, a skill and calling commensurate to virtuoso jazz musicians. Algorithmically speaking I do think that jazz is a harder climb. I don't think we have the algorithmic sophistication yet. I am not sure we ever will.

But it is not the purpose of algorithmic composition to play jazz. In fact algorithmic composition has quickly transcended any genre of music from classical counterpoint to gestural new music, both of which algorithmic composers have turned their hands to. Algorithmic composers will be best known for forging new areas of sound, sonic landscapes that cannot be approached with non-algorithmic approaches with some parts that cannot be played by human musicians.

If you go beyond this book and take up algorithmic composition I hope that you will at least consider mapping out new areas in musical experience. That potential is certainly there.



Can robots play jazz?

Kier Heyl drew the robots and the cover art.

Glossary

4/4 time - A time signature of 4/4 means count 4 (top number) quarter notes (bottom number) to each bar.
https://www.studybass.com/lessons/reading-music/time-signatures/

algorithm - A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
https://www.lexico.com/definition/algorithm

array - When referring to programming, an array is a group of related data values (called elements) that are grouped together.
https://www.computerhope.com/jargon/a/array.htm

articulation - Directions to a performer typically through symbols and icons on a musical score that indicate characteristics of the attack, duration, and decay.
http://dictionary.onmusic.org/terms/226-articulation

attack - The amount of time it takes for the envelop to reach the end of that first stage, usually the peak level.
https://theproaudiofiles.com/synthesis-101-envelope-parameters-uses/

Band-in-a-Box - Auto-accompaniment program for songwriting, education, and music production.
https://www.pgmusic.com/

bar - Lines drawn perpendicularly across the staff to divide it into measures. The term also means measure in common usage, but the bar is strictly the line itself, and not the measure it defines.
http://dictionary.onmusic.org/terms/4831-bar

cadence - A stylized close in music which divides the music into periods or brings it to a full conclusion.
http://dictionary.onmusic.org/terms/558-cadence

cartesian coordinates - Using Cartesian Coordinates we mark a point on a graph by how far along and how far up it is.
https://www.mathsisfun.com/data/cartesian-coordinates.html

Charlie Parker Omnibook - The world famous "Gold Standard" of jazz solo books.
https://www.jazzbooks.com/mm5/merchant.mvc?Screen=PROD&Product_code=OMNI-E

chords - The sounding of two or more notes (usually at least three) simultaneously.
http://dictionary.onmusic.org/terms/690-chord

church modes - Modes built from the notes in the C major scale (white keys on the piano).
https://bandnotes.info/tidbits/tidbits-feb.htm

clef - A symbol placed at the beginning of the staff to denote which notes are indicated by the lines and spaces.
http://dictionary.onmusic.org/terms/783-clef

clipboard - The clipboard, also known as pasteboard, is a special location in your computer or mobile device's memory that temporarily stores cut or copied text or other data.
https://www.computerhope.com/jargon/c/clipboar.htm

code - Code (short for source code) is a term used to describe text that is written using the protocol of a particular language by a computer programmer.
https://www.computerhope.com/jargon/c/code.htm

command prompt - Any prompt received in a command line interface that waits for user input before proceeding to the next step or performing the command entered.
https://www.computerhope.com/jargon/c/commprom.htm

conditional - A conditional statement is a set of rules performed if a certain condition is met. It is sometimes referred to as an If-Then statement, because IF a condition is met, THEN an action is performed.
https://www.computerhope.com/jargon/c/contstat.htm

copy and paste - Copy and paste or copy is the act of duplicating text, data, files, or disks, producing two or more of the same file or segments of data.
https://www.computerhope.com/jargon/c/copy.htm

counting numbers - Whole Numbers are simply the numbers 0, 1, 2, 3, 4, 5, ... (and so on). Counting Numbers are Whole Numbers, but without the zero.
https://www.mathsisfun.com/whole-numbers.html

cymbal - A percussion instrument made of a circular brass plate.
http://dictionary.onmusic.org/terms/1002-cymbal

database - Alternatively referred to as a databank or a datastore, and sometimes abbreviated as a DB, a database is a large quantity of indexed digital information.
https://www.computerhope.com/jargon/d/database.htm

data set - A structured collection of data.

DBMS - Short for database management system, DBMS is a software program that allows the user to create, manipulate, retrieve, and store information in a database.
https://www.computerhope.com/jargon/d/dbms.htm

decay - The amount of time it takes for the envelope to decrease to some specified sustain level.
https://theproaudiofiles.com/synthesis-101-envelope-parameters-uses/

decision tree - A decision tree can be used to visually and explicitly represent decisions and decision making.
https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

delimiter - A delimiter is one or more characters that separate text strings.
https://www.computerhope.com/jargon/d/delimite.htm

deterministic - In deterministic algorithm, for a given particular input, the computer will always produce the same output going through the same states but in case of non-deterministic algorithm, for the same input, the compiler may produce different output in different runs.
https://www.geeksforgeeks.org/difference-between-deterministic-and-non-deterministic-algorithms/

diatonic - Proceeding in the order of the octave based on five tones (steps) and two semitones (half steps).
http://dictionary.onmusic.org/terms/1070-diatonic

drum kit - A drummer in a rock or jazz band usually plays a "kit" (sometimes referred to as a drum set) or a specific group of untuned percussion instruments.
http://dictionary.onmusic.org/terms/1227-drum_kit
dynamics - Dynamics refers to the volume of a sound or note.
https://courses.lumenlearning.com/musicappreciation_with_theory/chapter/dynamics-and-dynamics-changes/

eighth note - A note having the time duration of one eighth of the time duration of a whole note.
http://dictionary.onmusic.org/terms/1281-eighth_note

encapsulation - Surrounding the target tone with chromatics until its resolution is desired.
https://truefire.com/theory-guitar-lessons/sequences-patterns-improvisation/chromatic-pattern--encapsulating-chords-encapsulation/v42019

enharmonic - The phenomenon that two separate pitch notations stand for the same sound. For example, the enharmonic spelling of F sharp is G flat. Both represent the same pitch frequency.
http://dictionary.onmusic.org/terms/1316-enharmonic

explode - In php the explode() function breaks a string into an array.
https://www.w3schools.com/php/func_string_explode.asp

factors - Factors are what we can multiply to get the number.
https://www.mathsisfun.com/numbers/factors-multiples.html

fifth interval - An interval of five diatonic degrees, counting the first and last degree, for example, a fifth above C would be G.
http://dictionary.onmusic.org/terms/1407-fifth

Finale - The sheet music program I used writing this book. When I say Finale I mean Finale or your sheet music program. Any program that imports and exports musicXML will work.
https://www.finalemusic.com/

flag - A flag is also a mark that indicates a certain event has taken place, or that an object is unusual in some way.
https://www.computerhope.com/jargon/f/flag.htm

for loop - A loop statement in programming that performs predefined tasks while or until a predetermined condition is met.
https://www.computerhope.com/jargon/f/for.htm

Fundamental theorem of arithmetic - Prime Factorization by Wouter Kager
http://www.few.vu.nl/~wkager/download/PFT.pdf

genre - Style or manner. In music, a unique category of composition with similar style, form, emotion, or subject.
http://dictionary.onmusic.org/terms/1567-genre

group theory - A group is a set combined with an operation.
https://www.mathsisfun.com/sets/groups-introduction.html

half note - A note that has half the duration of a whole note.
http://dictionary.onmusic.org/terms/1650-half_note

harmonic cadence - A harmonic cadence is a two-chord progression at the end of a phrase of music.
https://www.musikalessons.com/blog/2017/09/cadences/

heat death of the universe - When all the energy the in the cosmos is uniformly spread out, there is no more heat or free energy to fuel processes that consume energy, such as life.
https://phys.org/news/2015-09-fate-universeheat-death-big-rip.html

hello world - "Hello World" is a software program or script that introduces someone to a programming language.
https://www.computerhope.com/jargon/h/hello.htm

iREAL PRO - Auto accompaniment program for android, ios, and mac.
https://www.irealpro.com/

input file - A data file used as input for a computer program.

integer - A number with no fractional part (no decimals).
https://www.mathsisfun.com/definitions/integer.html

interval - The distance between two pitches.
http://dictionary.onmusic.org/terms/1792-interval

iteration - With computing, iteration describes going through a set of operations that deal with computer code.
https://www.computerhope.com/jargon/i/iteration.htm

key - A specific scale or series of notes defining a particular tonality.
http://dictionary.onmusic.org/terms/1857-key

kick drum - A pedal operated bass drum usually found as part of a drum kit (drum set) played by a single musician in a rock, jazz, or other popular style ensemble.
http://dictionary.onmusic.org/terms/1865-kick_drum

loop - A loop is a software program or script that repeats the same instructions or processes the same information over and over until receiving the order to stop.
https://www.computerhope.com/jargon/l/loop.htm

mariadb - MariaDB is a community-developed RDBMS (relational database management system).
https://www.computerhope.com/jargon/m/mariadb.htm

measure - American term, equivalent to the English term "bar ", signifying the smallest metrical divisions of a composition, containing a fixed number of beats , marked off by vertical lines on the staff.
http://dictionary.onmusic.org/terms/2121-measure

melody - A tune; a succession of tones comprised of mode, rhythm, and pitches so arranged as to achieve musical shape, being perceived as a unity by the mind.
http://dictionary.onmusic.org/terms/2138-melody

middle C - The name given to the note that has the pitch value of 261.63 Hz. It is the note on the ledger line halfway between the bass and treble clef on the great staff.
http://dictionary.onmusic.org/terms/2187-middle_c

motif - A short tune or musical figure that characterizes and unifies a composition.
http://dictionary.onmusic.org/terms/2253-motif

musicXML - MusicXML was designed from the ground up for sharing sheet music files.
https://www.musicxml.com/
mysql - Introduced in 1998, MySQL is an open source RDBMS.
https://www.computerhope.com/jargon/m/mysql.htm

naming convention - Descriptive file names are an important part of organizing, sharing, and keeping track of data files. Develop a naming convention based on elements that are important to the project.
https://libguides.princeton.edu/c.php?g=102546&p=930626

nested loop - A nested loop is a loop within a loop, an inner loop within the body of an outer one.
https://tutorialink.com/php/nesting-loops.php

note - A notational symbol used to represent the duration of a sound and, when placed on a music staff, to also indicate the pitch of the sound.
http://dictionary.onmusic.org/terms/2364-note

octave designation - When specifying a particular pitch precisely, we also need to know the register. middle C is C4. An octave higher than middle C is C5, and an octave lower than middle C is C3.
http://openmusictheory.com/pitches.html

on the one - In this video, legendary bassist, Bootsy Collins, explains how James Brown taught him how to play (or not play) on "the one" and take funk to the next level.
https://www.thelooploft.com/blogs/ryans-corner/bootsy-collins-explains-playing-on-the-one

output file - A file written by a computer program with the results or output of the computation.

parallel arrays - Multiple arrays of the same size such that i-th element of each array is closely related and all i-th elements together represent an object or entity.
https://www.geeksforgeeks.org/parallel-array/

percentile dice - Instructions for "Rolling Percentile (d100)".
http://dmsworkshop.com/2020/04/10/d100/

php - PHP is a popular general-purpose scripting language that is especially suited to web development.
https://www.php.net/

plain text - Plain text, Plain-text, or Plaintext is any text, text file, or document containing only text.
https://www.computerhope.com/jargon/p/plaintex.htm

polyrhythm - The use of several patterns or meters simultaneously, a technique used in 20th century compositions.
http://dictionary.onmusic.org/terms/2663-polyrhythm

positive integers - All integers greater that 0. The counting numbers.

prime factors - "Prime Factorization" is finding which prime numbers multiply together to make the original number.
https://www.mathsisfun.com/prime-factorization.html

prime number - A Prime Number is a whole number that cannot be made by multiplying other whole numbers.
https://www.mathsisfun.com/prime-composite-number.html

probability - How likely something is to happen.
https://www.mathsisfun.com/data/probability.html

pseudo code - Plain English that cannot be compiled or executed, but explains a resolution to a problem.
https://www.computerhope.com/jargon/p/pcode.htm

quantify - To measure or judge the size or amount of something.
https://dictionary.cambridge.org/us/dictionary/english/quantify

quarter note - A note having the time duration of one fourth of the time duration of a whole note.
http://dictionary.onmusic.org/terms/2751-quarter_note

query - With a database or search, a query is a field or option used to locate information within a database or another location.
https://www.computerhope.com/jargon/q/query.htm

random - The term random refers to any collection of data or information with no determined order, or is chosen in a way that is unknown beforehand.
https://www.computerhope.com/jargon/r/random.htm

release - The time it takes for the output to decrease to zero after the key is released or the sustain instruction ends.
https://theproaudiofiles.com/synthesis-101-envelope-parameters-uses/

rests - A symbol standing for a measured break in the sound with a defined duration.
http://dictionary.onmusic.org/terms/2879-rest

reverse engineering - Reverse engineering involves finding out how various functions in the code are built, what they do, and how each relates to and interacts with other code functions.
https://www.computerhope.com/jargon/r/reverse-engineering.htm

rhythmic cadence - A rhythmic cadence provides a similar function to their harmonic counterparts, except that the focus here is strictly on the rhythmic placement of chords as opposed to the harmonic notes. Metrically accented cadences are rhythm cadences where the final chord of a progression ends on a strong beat (a downbeat). Metrically unaccented cadences are ones that end on a weak beat.
https://www.musikalessons.com/blog/2017/09/cadences/

ring theory - A ring in the mathematical sense is a set S together with two binary operators + and *.
https://mathworld.wolfram.com/Ring.html

root - The tonic or fundamental note of a chord.
http://dictionary.onmusic.org/terms/2942-root

score - The entirety of the instrumental and vocal parts of a composition in written form, placed together on a page in staves placed one below the other.
http://dictionary.onmusic.org/terms/3036-score

search and replace - Alternatively referred to as Find and Replace or Replace is the act of finding text and replacing the found text with an alternative.
https://www.computerhope.com/jargon/r/replace.htm

sequence (math) - A sequence is an ordered list containing successive items, or functions for performing certain actions.
https://www.computerhope.com/jargon/s/sequence.htm

sequence (music) - A restatement of an idea or motif at a different pitch level from the original.
http://dictionary.onmusic.org/terms/3081-sequence

seventh chord - A chord consisting of a root note, the third above the root, the fifth above the root, and the seventh above the root.
http://dictionary.onmusic.org/terms/3097-seventh_chord

sixteenth note - A note having the time duration of one sixteenth of the time duration of a whole note.
http://dictionary.onmusic.org/terms/3197-sixteenth_note

snare drum - A drum common in orchestral, band, and jazz music with two drum heads. It is named after the "snares" or strings stretched across the lower drum head.
http://dictionary.onmusic.org/terms/3225-snare_drum

snippet - Alternatively referred to as a code snippet, a snippet is a small portion of text that is part of a larger set of programming code.
https://www.computerhope.com/jargon/s/snippet.htm

Sonny Rollins Omnibook - The Sonny Rollins Omnibook celebrates the bebop legend.
https://www.jazzbooks.com/mm5/merchant.mvc?Screen=PROD&Store_Code=JAJAZZ&Product_Code=SRO-BF&Category_Code=

SQL - Short for Structured Query Language, SQL, ... was developed by Dr. Edgar F. Codd at the IBM research center in 1974. Today, SQL has become the de facto standard database language.
https://www.computerhope.com/jargon/s/sql.htm

sustain - The level of output while a sustain instruction persists (held note). It is important to observe that the sustain parameter is a measure of level, not time. This stage can theoretically last indefinitely.
https://theproaudiofiles.com/synthesis-101-envelope-parameters-uses/

synth envelope - Envelopes can be used in synthesis to modulate literally anything in the signal path. Most often thought of as amplitude modulation with the parameters attack, sustain, decay, release.
https://theproaudiofiles.com/synthesis-101-envelope-parameters-uses/

terminal - The terminal is an interface that allows you to access the command line.
https://www.computerhope.com/jargon/t/terminal.htm

text editor - The term editor is commonly used to refer to a text editor, which is a software program that allows users to create or manipulate plain text.
https://www.computerhope.com/jargon/e/editor.htm

text file - A text file is a computer file that only contains text and has no special formatting.
https://www.computerhope.com/jargon/t/textfile.htm

timbre - The quality of a sound; that component of a tone that causes different instruments (for example a guitar and a violin) to sound different from each other while they are both playing the same note.
http://dictionary.onmusic.org/terms/3587-timbre

time signature - A symbol placed at the left side of the staff indicating the meter of the composition.
http://dictionary.onmusic.org/terms/3589-time_signature

tonality - The principal of organization of a composition around a tonic based upon a major or minor scale.
http://dictionary.onmusic.org/terms/3612-tonality

tone - The particular sound of an instrument or voice, as well as the performer's particular coloring of that sound.
http://dictionary.onmusic.org/terms/3614-tone

tonic - The note upon which a scale or key is based.
http://dictionary.onmusic.org/terms/3622-tonic

transcriber - The person who made the transcription.

transcription - Transcribing music means writing down what you hear when you listen to a song or piece.
https://www.musical-u.com/learn/how-do-you-transcribe-music/

triplet - Three notes of equal length that are to be performed in the duration of two notes of equal length.
http://dictionary.onmusic.org/terms/3690-triplet

unrolled loops - Loop unrolling is a loop transformation technique that helps to optimize the execution time of a program. We basically remove or reduce iterations.
https://www.geeksforgeeks.org/loop-unrolling/

variable - A variable is a named unit of data that is assigned a value.
https://www.computerhope.com/jargon/v/variable.htm

walking bass - In jazz, a walking bass usually moves by steps played on bass or piano, with each note usually having the duration of a quarter note.
http://dictionary.onmusic.org/terms/3906-walking_bass

whole note - The whole note has the longest note duration in modern music.
https://www.musictheory.net/lessons/11

XML - XML is a software- and hardware-independent tool for storing and transporting data.
https://www.w3schools.com/xml/xml_whatis.asp