Chapter 4: Sound and Music.

Now that we have color, graphics, and an understanding of variables, let's add sound and make some music. Basic concepts of the physics of sound and musical notation will be introduced. You will be able to translate a tune into frequencies and durations to have the computer synthesize a voice.

Sound Basics – Things you need to know about sound:

Sound is created by vibrating air striking your ear-drum. These vibrations are known as sound waves. When the air is vibrating quickly you will hear a high note and when the air is vibrating slowly you will hear a low note. The rate of the vibration is called frequency.

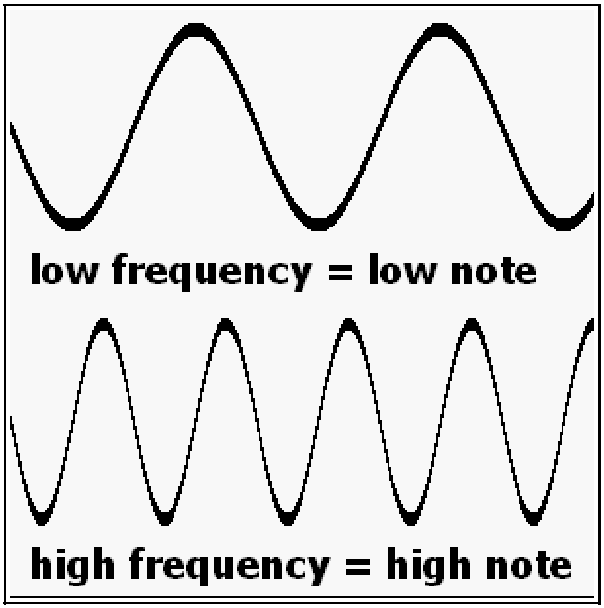


Illustration 9: Sound Waves

Frequency is measured in a unit called hertz (Hz). It represents how many cycles (ups and downs) a wave vibrates through in a second. A normal person can hear very low sounds at 20 Hz and very high sounds at 20,000 Hz. BASIC-256 can produce tones in the range of 50Hz to 7000Hz.

Another property of a sound is its length. Computers are very fast and can measure times accurately to a millisecond (ms). A millisecond (ms) is 1/1000 (one thousandths) of a second.

Let's make some sounds.

```
# sounds.kbs
sound 233, 1000
sound 466, 500
sound 233, 1000
```

Program 23: Play Three Individual Notes

You may have heard a clicking noise in your speakers between the notes played in the last example. This is caused by the computer creating the sound and needing to stop and think a millisecond or so. The *sound* statement also can be written using a list of frequencies and durations to smooth out the transition from one note to another.

In the program below, the first two values represent the frequency and duration of the first note. Once that is played the next two values are used to play the next note.

```
# soundslist.kbs
sound {233, 1000, 466, 500, 233, 1000}
```

Program 24: List of Sounds

This second sound program plays the same three tones for the same duration but the computer creates and plays all the sounds at once, making them smoother.


```
sound frequency, duration
sound {frequency1, duration1, frequency2,
      duration2 …}
sound numeric_array[]
```

The basic *sound* statement takes two arguments; (1) the frequency of the sound in Hz (cycles per second) and (2) the length of the tone in milliseconds (ms).

The second form of the sound statement uses a single list with curly braces to define the frequency and duration. This form can be confusing, be careful.

The third form of the sound statement uses an array containing frequencies and durations. Arrays are covered in a later chapter.

How do we get BASIC-256 to play a tune? The first thing we need to do is to convert the notes on a music staff to frequencies. Illustration 9 shows two octaves of music notes, their names, and the approximate frequency the note makes. In music you will also find a special mark called the rest. The rest means not to play anything for a certain duration. If you are using a list of sounds you can insert a rest by specifying a frequency of zero (0) and the needed duration for the silence.

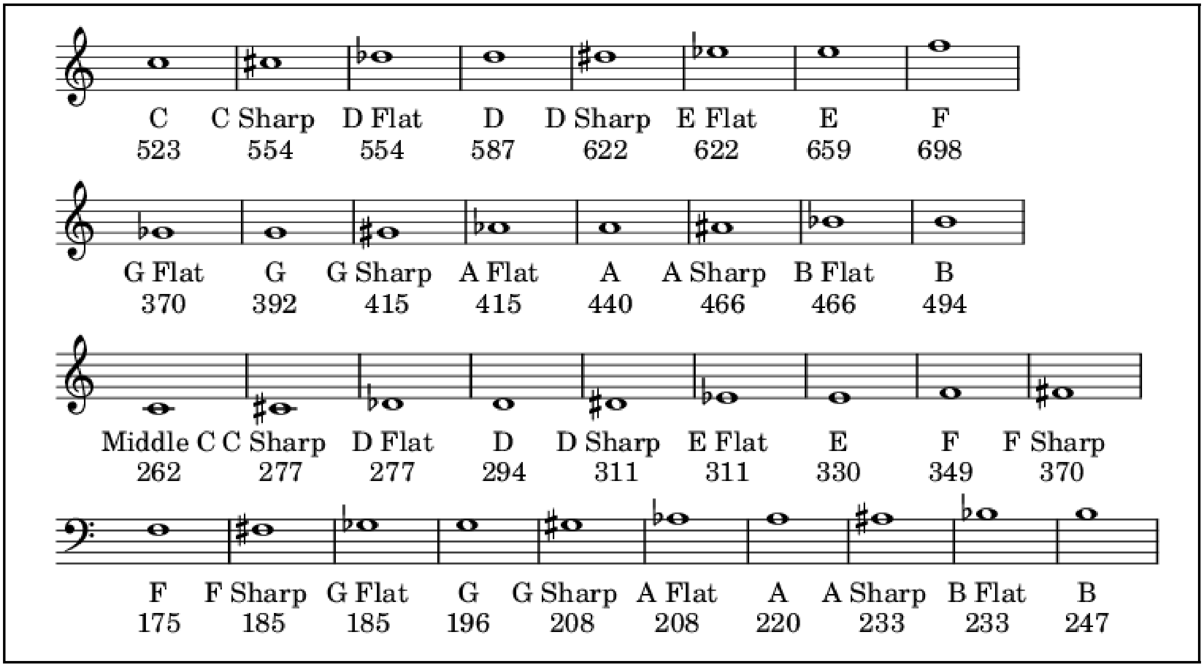


Illustration 10: Musical Notes

Take a little piece of music and then look up the frequency values for each of the notes. Why don't we have the computer play "Charge!". The music is in Illustration 11. You might notice that the high G in the music is not on the musical notes; if a note is not on the chart you can double (to make higher) or half (to make lower) the same note from one octave away.

Illustration 11: Charge!

Now that we have the frequencies we need the duration for each of the notes. Table 3 shows most of the common note and rest symbols, how long they are when compared to each other, and a few typical durations.

Duration in milliseconds (ms) can be calculated if you know the speed if the music in beats per minute (BPM) using Formula 1.

$$Note\ Duration = 1000 * 60 / Beats\ Per\ Minute * Relative\ Length$$

Formula 1: Calculating Note Duration

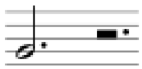
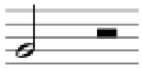
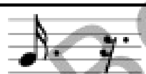
Note Name	Symbols for Note - Rest	Length in Beats	At 100 BPM	At 120 BPM	At 140 BPM
Dotted Whole		6.000	3600 ms	3000 ms	2571 ms
Whole		4.000	2400 ms	2000 ms	1714 ms
Dotted Half		3.000	1800 ms	1500 ms	1285 ms
Half		2.000	1200 ms	1000 ms	857 ms
Dotted Quarter		1.500	900 ms	750 ms	642 ms
Quarter		1.000	600 ms	500 ms	428 ms
Dotted Eighth		0.750	450 ms	375 ms	321 ms
Eighth		0.500	300 ms	250 ms	214 ms
Dotted Sixteenth		0.375	225 ms	187 ms	160 ms
Sixteenth		0.250	150 ms	125 ms	107 ms

Table 3: Musical Notes and Typical Durations

Now with the formula and table to calculate note durations, we can write the program to play "Charge!".

```
# charge.kbs - play charge
sound { 392, 375, 523, 375, 659, 375, 784, 250, 659,
250, 784, 250}
say "Charge!"
```

Program 25: Charge!

Instead of manually calculating the note durations, let's use a few variables to calculate and store the lengths for us. Using variables we could re-write the "Charge!" program using them to store the results of formulas to calculate note durations (Formula 1).

```
# charge2.kbs
# play charge - use variables
bpm = 120              # beats per minute
bms = 1000 * 60 / bpm          # ms per beat
dottedeighth = bms * .75
eighth = bms * .5
sound { 392, dottedeighth, 523, dottedeighth, 659,
dottedeighth, 784, eighth, 659, eighth, 784, eighth }
say "Charge!"
```

Program 26: Charge! with Variables

Big Program	For this chapter's big program let's take a piece of music by J.S. Bach and write a program to play it. The musical score is a part of J.S. Bach's Little Fuge in G.

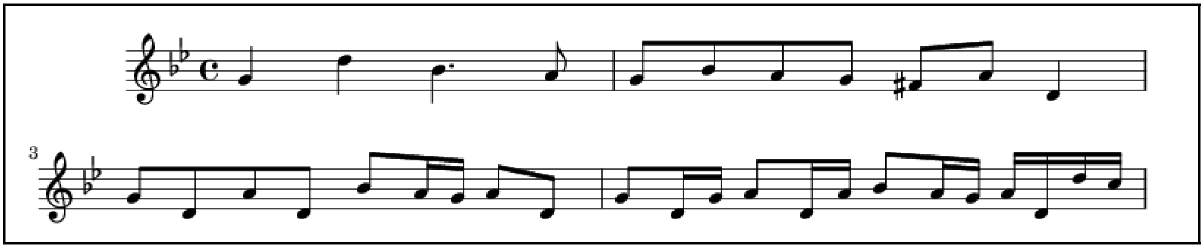


Illustration 12: First Four Measures of J.S. Bach's Little Fuge in G

```
# littlefuge.kbs
# Music by J.S.Bach - XVIII Fuge in G moll.

tempo = 100 # beats per minute
milimin = 1000 * 60 # miliseconds in a minute
q = milimin / tempo # quarter note is a beat
h = q * 2 # half note (2 quarters)
e = q / 2 # eight note (1/2 quarter)
s = q / 4 # sixteenth note (1/4 quarter)
de = e + s # dotted eight - eight + 16th
dq = q + e # doted quarter - quarter + eight

sound {392, q, 587, q, 466, dq, 440, e, 392, e, 466,
e, 440, e, 392, e, 370, e, 440, e, 294, q, 392, e,
294, e, 440, e, 294, e, 466, e, 440, s, 392, s, 440,
e, 294, e, 392, e, 294, s, 392, s, 440, e, 294, s,
440, s, 466, e, 440, s, 392, s, 440, s, 294, s}
```

Program 27: Big Program - Little Fuge in G

Exercises:

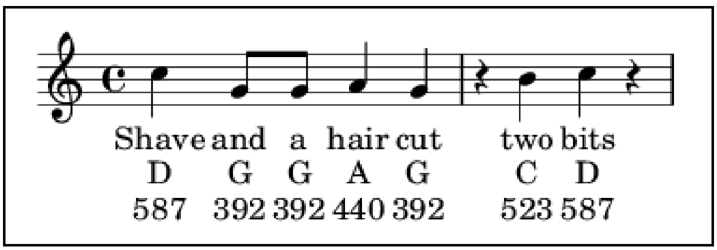
Word Search

d	j	r	a	h	e	r	t	z	q	y	t	x
n	a	v	a	r	i	a	b	l	e	l	z	s
o	s	h	a	l	f	n	g	k	j	u	e	x
c	s	s	h	o	r	t	c	u	t	c	g	j
e	i	e	h	t	h	g	i	e	a	h	i	n
s	g	t	u	r	l	s	l	r	t	b	k	x
i	n	a	t	y	f	i	b	n	d	e	d	t
l	m	r	s	a	i	x	e	n	e	x	l	u
l	e	b	y	c	n	e	u	q	e	r	f	i
i	n	i	b	q	t	o	e	v	a	t	c	o
m	t	v	z	x	s	j	w	h	o	l	e	b
m	u	s	i	c	r	e	t	r	a	u	q	a
i	j	s	q	s	e	y	t	e	t	o	n	t

braces, eighth, frequency, half, hertz, millisecond, music, note, octave, quarter, shortcut, sixteenth, sound, vibrate, whole

Problems

1. Write a program using a single sound statement to play "Shave and a Hair Cut". Remember you must include the quarter rests in the second measure in your sound with a frequency of zero and the duration of a quarter note.



2. Type the sound statement below and insert the variable assignments before it to play "Row Row Row your Boat". The variables c, d, e, f, g, and cc should contain the frequency of the notes of the tune. The variable n4 should contain the length in milliseconds of a quarter note; n2 twice n4, and n8 one half of n4.

```
sound {c,n4+n8, c,n4+n8, c,n4, d,n8, e,n4+n8,
e,n4, d,n8, e,n4, f,n8, g,n2+n4, cc,n8, cc,n8,
cc,n8, g,n8, g,n8, g,n8, e,n8, e,n8, e,n8, c,n8,
c,n8, c,n8, g,n4, f,n8, d,n4, e,n8, c,n2+n4}
```