

# CSIS 10C

# Lab 5B

# Music Machine

In this lab we will explore the development of a music player capable of receiving input of song notes, in the form of a string, such as "ccggaag" for the first phrase of the song "Twinkle Twinkle Little Star" and playing it back to the user over the speaker.

The following circuit is already coded up (with the code shown at left) in Tinkercad at this [link](#). You may, of course, build your own version on a real Arduino -- the sound will be better!

```
// Sound.ino

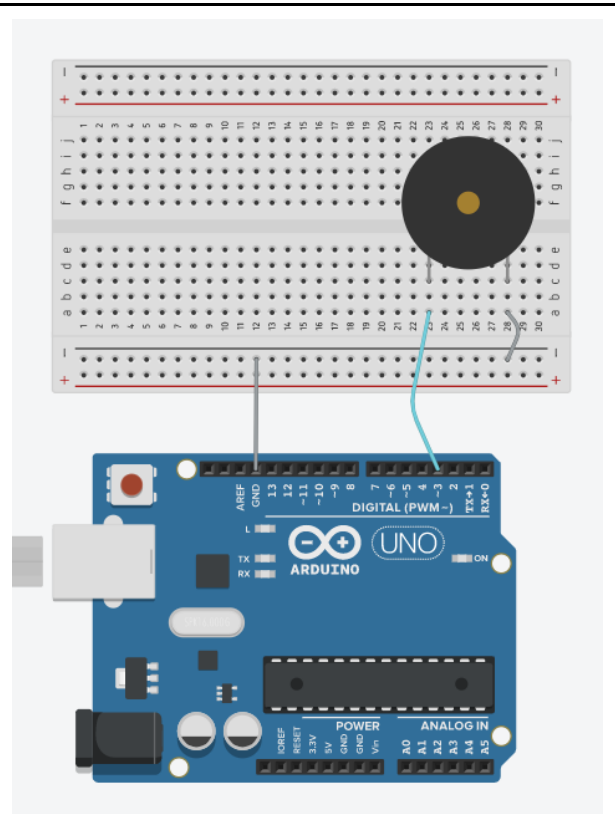
int speakerPin = 3;

String allNotes = "cCdDefFgGaAb";

int allPitches[] =
{261,277,294,311,330,349,370,392,415,
440,466,494};

void setup() {
  Serial.begin(9600);
  // start here
}

void loop() {
}
```



## Part 1: Twinkle Twinkle

1. To begin with, we will start by exploring how to make a simple tone play on the Arduino. The command to do this is **tone** (to start a tone) and **noTone** (to end the tone). Between these two statements, we add a delay so that we can hear the tone for a certain amount of time (200 ms or 1/5 of a second). Add this code to your **setup** function after "//start here".

```
tone(speakerPin,261);
delay(200);
noTone(speakerPin);
```

We are sending a pitch of 261 Hz to the speaker. On Tinkercad you will hear the tone kind of breaking up (at least that's what happens on my computer) probably because the computer system is being interrupted by other processes running on the computer. On the real Arduino, there is nothing else going on, so the tone sounds clearly without interruption.

2. You'll notice there is a string and an array defined at the top of the program:

```
String allNotes = "cCdDefFgGaAb";

int allPitches[] = {261, 277, 294, 311, 330, 349, 370, 392, 415, 440, 466, 494};
int allPitches[] = {C4, CS4, D4, DS4, E4, F4, FS4, D4, GS4, A4, AS4, B4};
```

These two define a series of pitches representing the musical notes of the scale. We just played 261 Hz, which represents the note called "middle c." And this corresponds to the number in the `allPitches` array, and the first letter of the `allNotes` string.

Notice there are uppercase and lowercase letters ... the uppercase letters like "C" represents the "sharp" note that is 1/2 step higher than the note or pitch for the lowercase letter.

3. To make sure you understand this, try playing the first four notes of "Twinkle Twinkle," in other words "c", "c", "g" "g". Do this by stringing notes one after the other in setup. Add a short, 10 or 20 msec delay between notes so you can clearly make out two notes that follow each other.

In other words,

```
tone(speakerPin, 261);    // "c"
delay(190);
noTone(speakerPin);
delay(10);    // brief pause between notes
tone(speakerPin, 261);    // "c"
delay(190);
noTone(speakerPin);
delay(10);    // brief pause between notes
// continue with "g" and "g"
```

## Part 2: Connecting notes to pitches

It's not terribly convenient to list each note separately. And to have to connect the pitch to the note ourselves. We'd like a way to streamline this process so a user can enter notes ("abc", etc) and have the Arduino play the corresponding pitches (440, 494, 261). To accomplish this, we will define two functions that work the the `allNotes` string and the `allPitches` array.

4. Add the following two functions to your program, above `setup()` and after the place where the `allPitches` array is defined:


```
/* isNote()
 * char note: The letter that represents the note
 * returns: true (1) if the note is found in the allNotes String
 */
bool isNote(char n){
    return (allNotes.indexOf(n) > -1);
}

/* getPitch()
 * char note: The letter that represents the note
 * returns: The note's frequency
 */
int getPitch(char note) {
    if (allNotes.indexOf(note) > -1)
        return allPitches[allNotes.indexOf(note)];
    else return 0;
}
```

The first function uses the string `indexOf` function to tell us what position in the `allNotes` array we can find a certain character, like the letter 'a'.

To see how these work, try adding these two lines to `setup` right after `Serial.begin(9600)`

```
Serial.println(isNote('a'));
Serial.println(isNote('x'));
```

When you run your program, it should print a 1 or 0 (1 meaning true, 0 meaning false) on whether the argument (passed to `char n`) is in the `allNotes` string. The output goes to the Serial Monitor (make sure you bring that up by clicking on the  Serial Monitor symbol in Tinkercad).

When you run the program, you should see the following output

```
1
0
```

The 1 means the letter 'a' IS found in the `allNotes` string. The 0 means the letter 'x' is NOT found in the `allNotes` string.

5. Next let's look at the `getPitch` function. Add these lines to your code, right below the previous ones you added in the previous step.

```
Serial.println(getPitch('a'));
Serial.println(getPitch('x'));
```

When you run your program again you should see the numbers

```
440  
0
```

added to your Serial Monitor output. The 440 indicates that the pitch we should play for a letter 'a'. Notice how the function works -- it finds the index 9 (starting from 0) in the allNotes string where the letter 'a' is located. It then uses that index to look up the corresponding position in the allPitches array, and returns that value to the caller. The 0 again indicates the letter 'x' is not in the allNotes string.

6. Finally, let's rewrite your "Twinkle Twinkle" code to use the getPitch function. Instead of saying

```
tone(speakerPin, 261);
```

revise the code to use

```
tone(speakerPin, getPitch('c'));
```

and do the same for the other 3 notes.

7. Once you understand how this works, you can change your setup function to the following:

```
void setup() {  
  Serial.begin(9600);  
  String scale = "cdefgabagfedc";  
  for (int i=0; i<scale.length(); i++) {  
    tone(speakerPin, getPitch(scale[i]));  
    delay(200);  
  }  
  noTone(speakerPin);  
  Serial.println("Type your song and press enter to  
    hear it play!");  
}
```

Spend a moment observing how this works. We are creating a string for the musical scale, then using a for loop to visit every note in the string, and the getNote function to turn that note (letter) into a pitch.

## Part 3: Music Machine!

Now that you can play a scale it's time to play a melody. The first thing we must do is read the serial input for the melody. A melody will be input all at once like this:

```
agfgaaa
```

The melody ends with the newline ('\n') character. When you read the newline character you should stop reading input and play the melody.

8. To achieve the input step, please modify your loop function to use the following code to (repeatedly) read a string in from the user and print the notes (pitches) out to the screen.

```
void loop() {  
  bool done = false;  
  while (Serial.available()>0) {  
    String song = Serial.readStringUntil('\n');  
    Serial.println(song);  
    for (int k=0; k<song.length(); k++){  
      Serial.print(getPitch(song[k]));  
      Serial.print(' ');  
    }  
    Serial.println();  
  }  
}
```

When you run the program now, after playing the opening scale, it asks the user to enter a melody, and prints the notes (and pitches) to the screen. The next step is to add code to turn this into a melody we can hear through the speaker.

9. The following table illustrates the notes we've already coded into our machine as allNotes and allPitches. **However, there are also two more symbols added at the end.** Check those out:

Note	Code Letter	Action
Middle C	c	261 Hz
C#	C	277 Hz
D	d	294 Hz
D#	D	311 Hz
E	e	330 Hz
F	f	349 Hz
F#	F	370 Hz
G	g	392 Hz
G#	G	415 Hz
A	a	440 Hz
A#	A	466 Hz
B	b	494 Hz
	-	Stretch last note
	(space)	Rest (no sound)

The extra symbols include a '-' for a "stretch" (holding the note continuously into the next interval) and a musical "rest" (no sound at all) when a gap in the sound is needed.

10. Here's where it gets interesting. The way your machine will work is to translate the notes into music as follows:

```
input: aaa
produces:
  440Hz (190ms) then quiet (10ms)
  440Hz (190ms) then quiet (10ms)
  440Hz (190ms) then quiet (10ms)
```

```
input: a--
produces:
  440Hz (200ms)
  440Hz (200ms)
  440Hz (200ms)
```

Note the difference between `aaa` (which includes a small gap of 10 ms between notes) and `"a--"` (which essentially holds the 440 Hz pitch continuously for 600 ms).

The trick is setting up your loop to process the notes, but also to keep track of "stretch" symbols which imply we have to change the behavior and play the last note again with no gap between them.

Also, a space in the input should just play 200ms of silence.

11. Now write the code to do the steps described above. Here are some test songs you can try:

```
Mary Had a Little Lamb:
agfgaaa ggg aaa agfgaaaaggagf

Yankee Doodle:
ffgafag ffgaf-e- ffgaAagfecdef-f-
```

12. Here are some hints you can use:

- First, simply change the notes to pitches and send them to the speaker using `tone` and `notone` commands.
- After you can hear the pitches, modify your program to handle `'-'` and `' '` characters in your song string
  - you should be able to enter `aacceec-` and hear the song
  - you should be able to hear the difference between `aaa` (3 different notes) and `a--` (one long note)
- HINTS on how to handle `'-'` properly
  - Nice idea but not quite: if you get a `'-'` play the `k-1`'th note
    - doesn't work because you already have a 10ms delay from previous note

- Better: Play a note, when go next k value that's when you decide to send a noTone (if a new note) or if it's a '-' keep the tone running another interval.
- MORE HINTS on handling '-'
  - for (int k = 0; k < song.length(); k++)
    - if the note at location k is NOT a '-'
      - noTone and delay for 10 ms, THEN tone the pitch of the note for 190 ms
    - else // it's a '-'
      - delay(200)

13. Last Challenge! See if you can play this on your Arduino -

[Imperial March Ringtone OFFICIAL](#)

Have fun with this! When you are finished, turn in your code to the farthest point you got to.