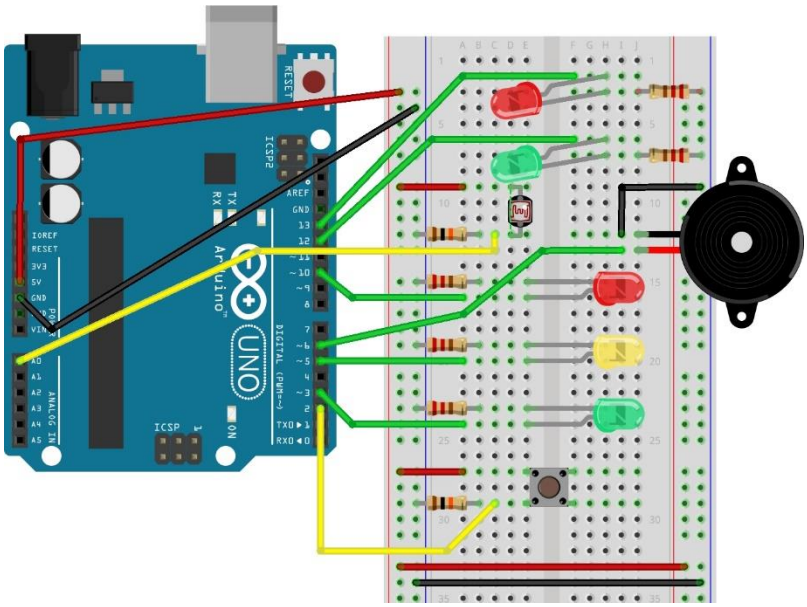
Arduino – Programmierpraxis mit Schall

Aufgabe 1) Dir ist sicher schon aufgefallen, dass die meisten Verkehrsampeln auch einen akustischen Signalgeber haben, der eine bestimmte Tonfolge erzeugt, wenn die Ampel für den fließenden Verkehr „grün“ zeigt, und eine andere Tonfolge, wenn die Ampel „rot“ zeigt. Dies hat den Sinn, dass auch sehbehinderte Fußgänger erkennen können sollen, wann sie die Fahrbahn überqueren dürfen.

In unserem ersten Projekt zur Schallerzeugung mit dem Arduino-Board wollen wir die bewährte „Ampelschaltung“ um diese Funktionalität erweitern:

Ergänze zunächst die dir zur Verfügung stehende Schaltung durch einen (großen schwarzen) Piezo-Lautsprecher – nebenstehend ist eine mögliche derartige Erweiterung für die Ampelschaltung mit Fußgängerampel abgebildet:



Verbinde dazu einen der beiden Kontakte des Piezo-Lautsprechers mit einem der freien digitalen Steckkontakte (z.B., wie abgebildet, mit dem Kontakt Nummer ~6).

Den anderen Kontakt verbinde direkt mit „Erde“ („GND“) – und schon kannst du dem Lautsprecher mit dem von der LED-Programmierung bekannten Befehl `digitalWrite` ein „Knacken“ oder „Tikken“ entlocken – lies‘ dazu bitte auch die **Erklärungen in der Informationsdatei ST_I_09Arduino_Praxis_Schall** nach.

Erweitere nun dein Programm zur Ampelsteuerung so, dass die „Töne“ des „tickenden“ Piezo-Lautsprechers unterschiedlich „klingen“, je nachdem, ob die Ampel „grün“ oder „rot“ zeigt – du kannst den Piezo-Lautsprecher aber zusätzlich auch noch so programmieren, dass ein eigenes Signal ertönt, bevor die Fußgängerampel von „grün“ wieder auf „rot“ schaltet.

Experimentiere dabei mit unterschiedlichen Parameterwerten beim `delay`- bzw. beim `delayMicroseconds` -Befehl. Fällt dir auf, dass die Töne beim „Ticken“ umso höher klingen, je kürzer die „Verzögerung“ durch einen der beiden `delay`-Befehle ist? Ändert sich bei unterschiedlichen Werten für die Verzögerung auch die Dauer des „Tickens“?

Aufgabe 2) In den bisherigen Programmieranregungen wurde die „Ampelschaltung“ auch als „Morseapparat“ programmiert, wobei die Morsesignale durch langes oder kurzes Aufleuchten einer LED repräsentiert waren. Tatsächlich wurden Morseapparate aber so gebaut, dass die Morsesignale durch lang oder kurz anhaltende Töne codiert wurden – historische Morseapparate haben also nicht „optisch“, sondern „akustisch“ funktioniert – und das kannst du mit der um den Piezo-Lautsprecher erweiterten „Ampelschaltung“ jetzt auch programmieren.

Verändere eines deiner codierten Morseprogramme so, dass die Unterprogramme `shortSignal` bzw. `longSignal` nicht ein kurzes oder ein langes Aufleuchten einer der Leuchtdioden, sondern ein kurz anhaltendes oder ein lang anhaltendes „Ticken“ des Piezo-Lautsprechers bewirken.

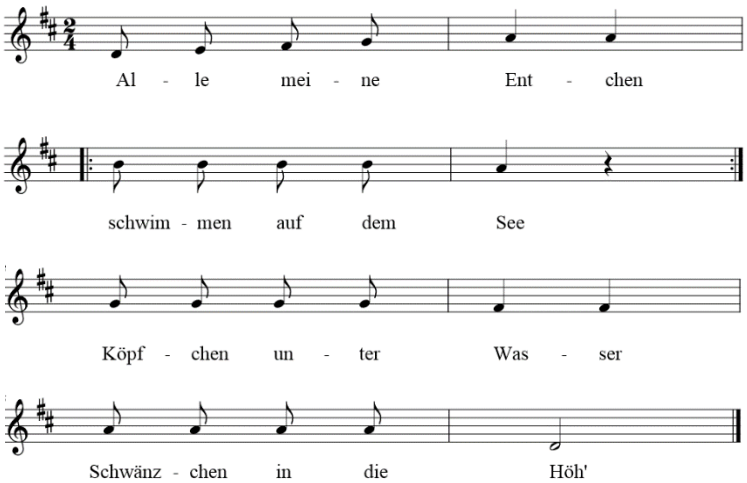
Ein Hinweis: Mit dem Befehl `digitalWrite` programmierst du ein langes „Ticken“ am einfachsten, indem du mehrere „Ticks“ knapp hintereinander erzeugst (du kannst aber auch eine zweite Version mit `tone`/`noTone` programmieren, beachte dazu den Hinweis zu Beginn von Aufgabe 3)… ☺)!

Teste sodann den „akustischen“ Morseapparat, indem du die gemorste Nachricht von jemandem anderen decodieren lässt – das ist ohne Übung nämlich gar nicht so einfach!

Aufgabe 3) In der **Informationsdatei** **ST_I_09Arduino_Praxis_Schall** kannst du auch erfahren, mit welchen Befehlen du einem Piezo-Lautsprecher Töne einer bestimmten Tonhöhe (Frequenz) und Dauer entlocken kannst, und wie in der Musik **Töne durch Noten codiert** werden – wenn du dies noch nicht gelesen hast, dann informiere dich bitte jetzt auf den Seiten zwei bis vier von dieser Informationsdatei.

Vielleicht überrascht es dich ja, dass es nicht nur in der Technik (z.B. beim Morsen oder beim Programmieren) Codes gibt?

Übe dich jedenfalls zuerst in der **Decodierung** des nebenstehenden **Notenblattes**, das das bekannte Kinderlied „Alle meine Entchen“ codiert.



Codiere sodann ein Arduino-**Programm** so, dass dieses Kinderlied vom Piezo-Lautsprecher „gespielt“ wird. Am besten codierst du zunächst jeden der 22 Töne (samt der einen Pause) „einzeln“, d.h. ähnlich zum dritten Codefragment auf Seite 2 der Informationsdatei **ST_I_09Arduino_Praxis_Schall.** Dadurch kannst du mit den Befehlen `tone` und `noTone` vertraut werden – experimentiere dabei auch, wie lang eine ganze Note dauern soll, damit das Lied vertraut klingt...

Noch ein Tipp: Es empfiehlt sich, auch nach dem `noTone` -Befehl eine kurze „Pause“ mit dem `delay`-Befehl einzufügen.

Aufgabe 4) Vermutlich war das Codieren des Programms in Aufgabe 3) recht mühsam. In der **Informationsdatei ST_I_09Arduino_Praxis_Schall** ist ab Seite 5 auch erklärt, wie der Programmcode mit Hilfe von Feldvariablen (Arrays) und einer Programmschleife wesentlich kompakter verfasst werden kann. Der Code für den Anfang von „Alle meine Entchen" kann damit beispielsweise so geschrieben werden:

```
int piezoPin;
int frequenzen[] = {297,330,352,396,440,440};
int durations[] = {200,200,200,200,400,400};
int arrayLen = sizeof(frequenzen)/sizeof(int);

void setup() {
  piezoPin = 6;
  pinMode(piezoPin, OUTPUT);
}

void loop() {
  int index;

  index = 0;
  while(index < arrayLen){
    tone(6, frequenzen[index]);
    delay(durations[index]);
    noTone(6);
    delay(200);
    index = index + 1;
  }
}
```

Dabei tritt auch etwas Neues auf: In der markierten Programmzeile wird die Anzahl der Werte, die in der Feldvariablen gespeichert werden (die „Länge" der Feldvariablen) „automatisch" berechnet:

`sizeof(frequenzen)` bestimmt, wieviel Speicher die Feldvariable namens **`frequenzen`**, in der Werte vom Datentyp Integer (`int`) gespeichter werden, insgesamt benötigt.

Dividiert man diese Zahl durch `sizeof(int)`, d.h. durch die Speichergröße, die für einen `int`-Wert benötigt wird, so ist das Ergebnis dieser Division die Anzahl der in der Feldvariablen gespeicherten (`int`-) Werte – und diese Anzahl wird in der Variablen namens **`arrayLen`** gemerkt.

a) Vervollständige das angegebene Programmfragment, sodass das gesamte Lied „Alle meine Entchen" gespielt wird, und teste das Programm.

Künstler spielen die Noten eines Musikstückes manchmal nicht der Reihe nach – sie improvisieren, indem sie Noten auslassen oder indem sie die Reihenfolge der Noten verändern. Wir wollen ein paar Beispiele programmieren und uns anhören, wie dann „Alle meine Entchen" auch klingen kann:

b) Verändere das Programm zu „Alle meine Entchen" so, dass das Lied von hinten nach vorne gespielt wird, d.h. die Variable **`counter`** zählt nicht von `0` bis **`arrayLen - 1`** nach oben sondern von **`arrayLen - 1`** bis `0` nach unten (verwende dazu **`counter = counter - 1`**). Teste das Programm.

c) Verändere das Programm zu „Alle meine Entchen" so, dass nur jede zweite Note gespielt wird. Teste das Programm.

d) Verändere das Programm zu „Alle meine Entchen" so, dass zuerst alle Noten auf ungeraden Indizes, danach alle Noten auf geraden Indizes gespielt werden. Teste das Programm.

e) Verändere das Programm zu „Alle meine Entchen" so, dass zuerst alle Noten auf ungeraden Indizes von vorne nach hinten, danach alle Noten auf geraden Indizes von hinten nach vorne gespielt werden. Teste das Programm.

f) Verändere… …fallen dir noch weitere, vielleicht originellere Möglichkeiten ein, „Alle meine Entchen" zu improvisieren?