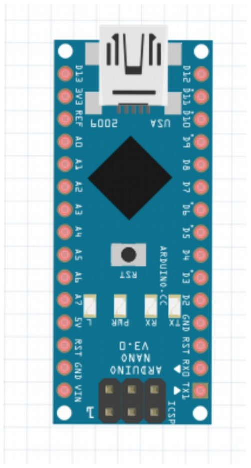
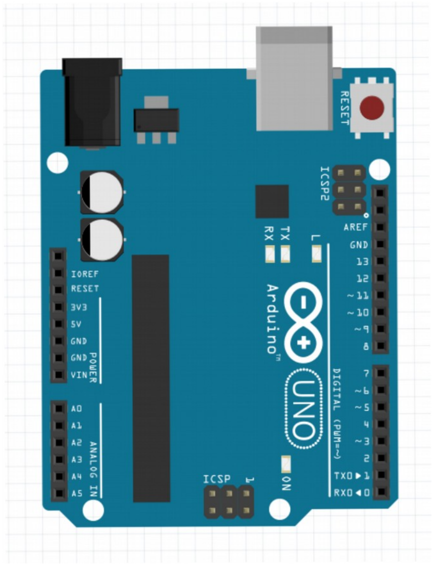
Playing multisound melodies on an Arduino

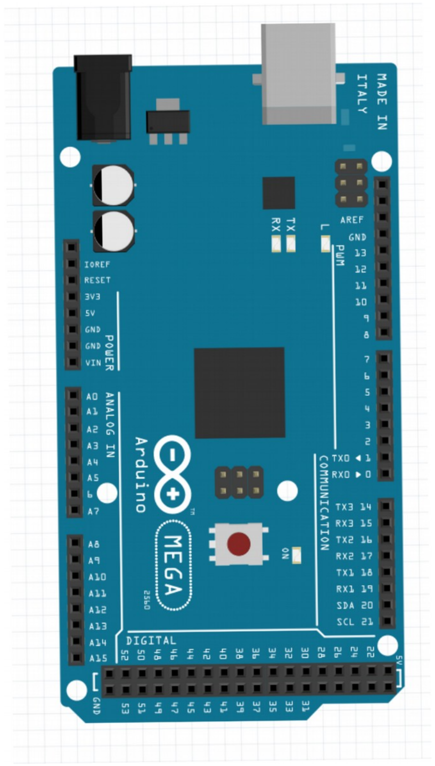
The Arduino is a small board with an ATMEL microcontroller.



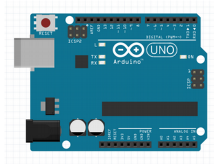
Nano Rev3
ATMEL 328P
32kB + 2 KB



Uno Rev3
ATMEL 328P
32kB + 2 KB

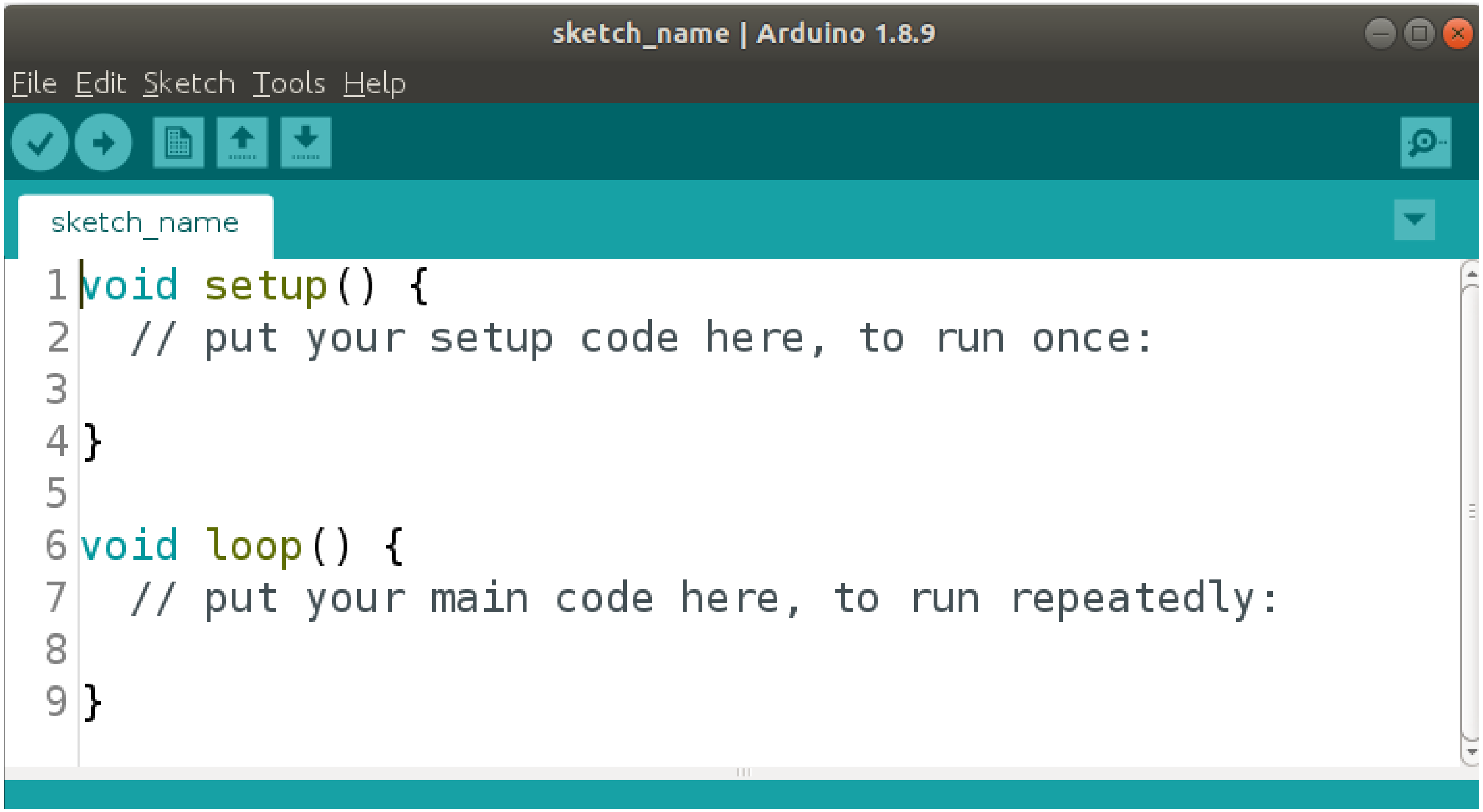


Arduino Mega 256 Rev3
ATMEL 2560P
256kB + 8 KB



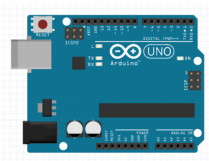
E.G. Hoffmann, Kiel / Germany

Starting the Arduino – IDE



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Allows programming close to modern C++.
Written on a PC. Upload per USB to the controller.

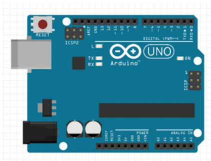


Many Arduino-libraries are available.
The most important for this project is
Brett Hagmans Tone-Library.

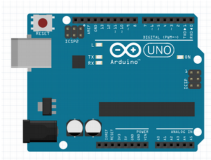
It uses the timers of the controller to **toggle**
(interrupt controlled) at arbitrary output ports
the voltage **between 0V and 5V.**

That’s the way to generate **squarewave** output
of wanted **frequencies** and **durations**.

Since the **MEGA 328** has three **timers**
it is possible to generate **up to three**
independent tone-outputs simultaneously.

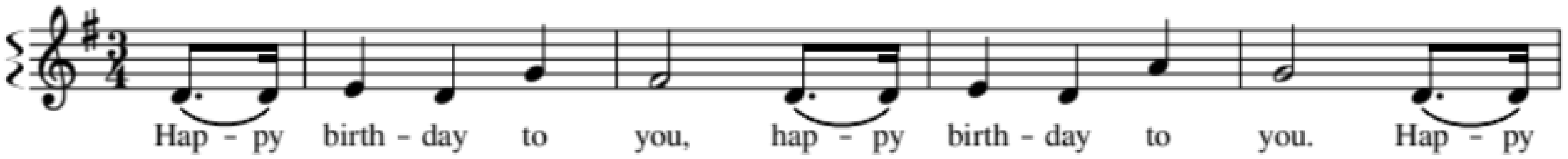


Outline

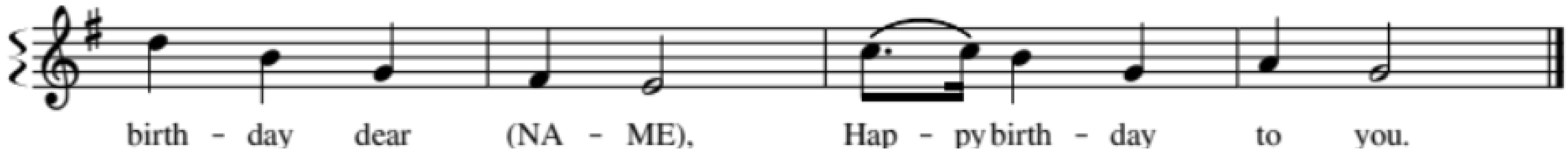


I. My Tone-notation

Let's start with one single voice and with coding melodies according to the common sheet music.


Hap - py birth - day to you, hap - py birth - day to you. Hap - py

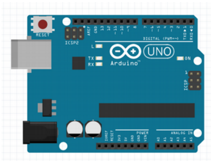
Coding: "D48.D46 E44 D44 G44 FS42 D48.D46 E44 D44 A44 G42 D44.D46 "



Coding: "D54 H44 G44 FS44 E42 C58.C56 H44 G44 A44 G42 "

Separate notes are separated by at least one space-character ' '.

Connected notes are not separated.

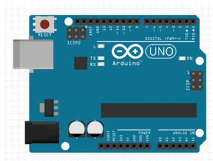


The first two notes are not separated **D48.D46** (D4 3/16 D4 1/16) .
For pauses in melodies we use the character **P**; **P4** is a quarter pause.

A note is syntactically:

```
note-item     ::= note | pause | |: | :| | ! .
pause         ::= P duration .
note          ::= note-letter octave duration .
note-letter   ::= A | B | C | D | E | F | G | H .
octave        ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 .
duration      ::= number appendix .
number        ::= 1 | 2 | 3 | 4 | 6 | 8 .  // 1/1, 1/2, 1/32, 1/4, 1/16, 1/8
appendix      ::= nothing | . | T .        // NOOP, duration *= 1.5, duration *= 2/3
```

For separated notes the duration will be reduced by about 5%
and a pause of just these 5% will be appended.



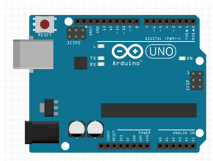
The first two notes are not separated **D48.D46** (D4 3/16 D4 1/16) .
For pauses in melodies we use the character **P**; **P4** is a quarter pause.

A note is syntactically:

```
note-item     ::= note | pause | |: | :| | ! .
pause         ::= P duration .
note          ::= note-letter octave duration .
note-letter   ::= A | B | C | D | E | F | G | H .
octave        ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 .
duration      ::= number appendix .
number        ::= 1 | 2 | 3 | 4 | 6 | 8 .  // 1/1, 1/2, 1/32, 1/4, 1/16, 1/8
appendix      ::= nothing | . | T .        // NOOP, duration *= 1.5, duration *= 2/3
```

For separated notes the duration will be reduced by about 5%
and a pause of just these 5% will be appended.

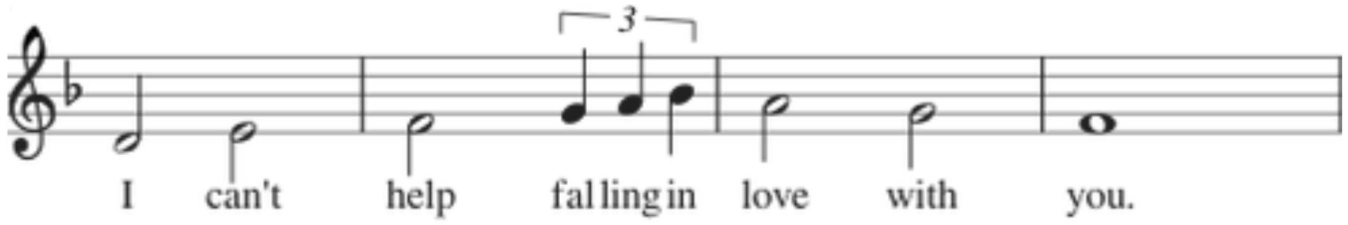
But what is with the **T** ?



But what is with the **T** ?

The best explanation I found is from **Elvis Presley**.

In his song "I can't help falling in love with you" he uses the **Triole**.
That means 3 notes in the time of 2 indicated by a bow with a 3 indication.



Coding:

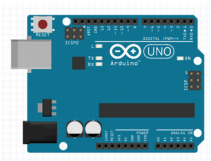
```
D42 E42      F42 G44T A44T B44T A42 G42      F41
```

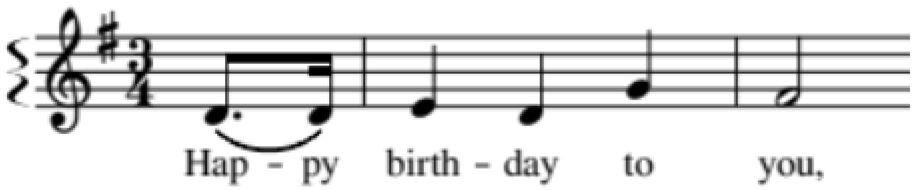
G44T A44T B44T: total duration = two quarter notes = one half note

II. Playing single-tone melodies

The **Tone.h** of **Brett Hagmans Tone-Library.**

```
class Tone {                                        // Tone.h
  public:                     // only the standard  constructor
    void begin(uint8_t tonePin);
    bool isPlaying();
    void play( uint16_t frequency,
               uint32_t duration = 0);
    void stop();
  private:
    static uint8_t _tone_pin_count;
    uint8_t _pin;
    int8_t _timer;
};
```

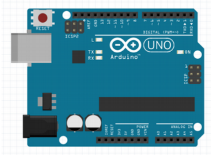


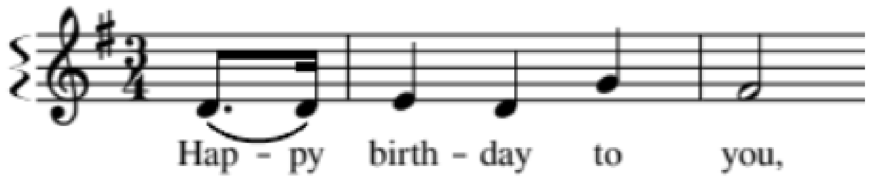


Done with Brett Hagmans Tone-library

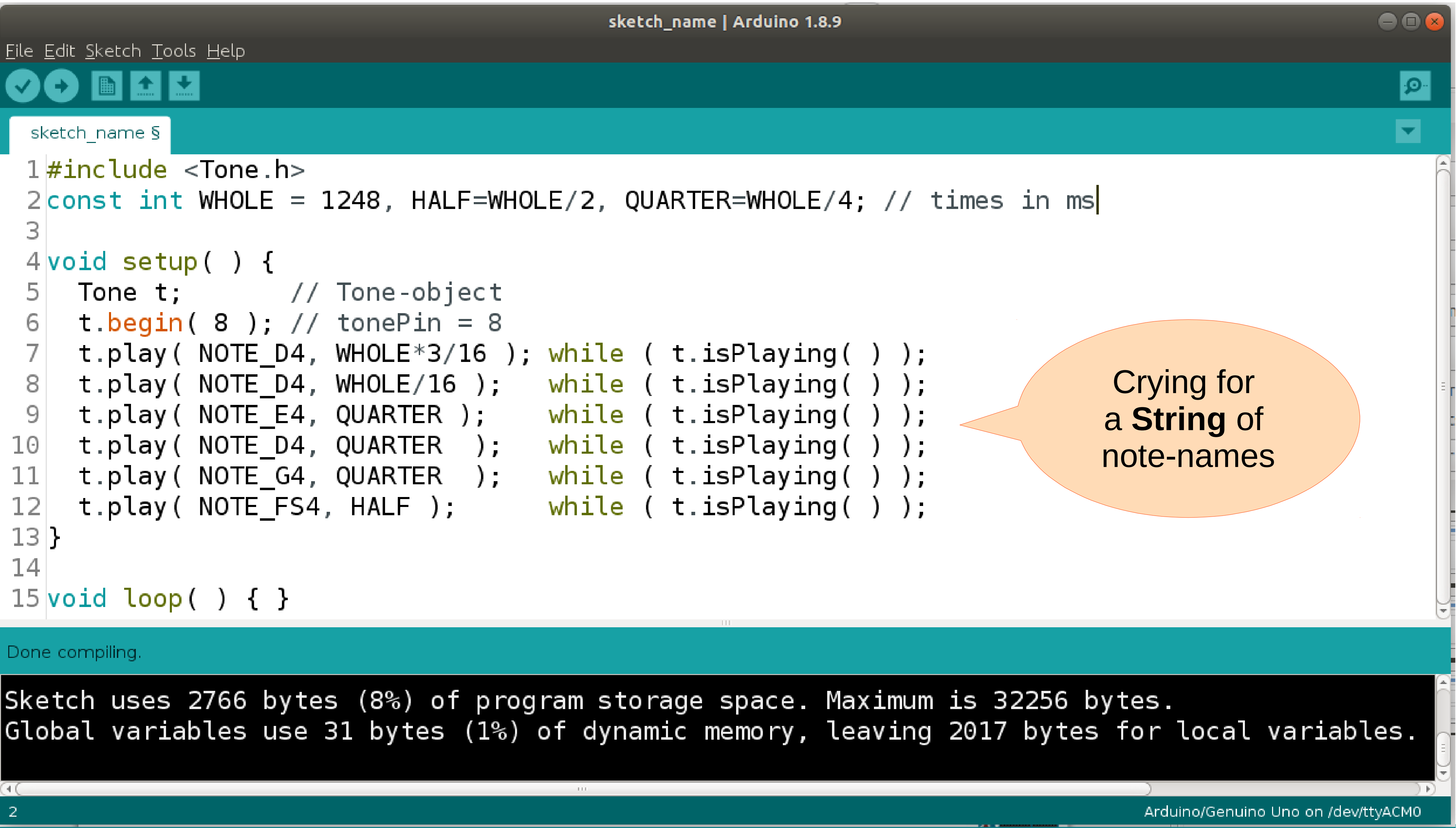
Coding: “D48.D46 E44 D44 G44 FS42 “

```
#include <Tone.h>
const int WHOLE = 1248, HALF=WHOLE/2, QUARTER=WHOLE/4;

void setup( ) {
  Tone t;           // Tone-object
  t.begin( 8 );    // tonePin = 8
  t.play( NOTE_D4, WHOLE*3/16 ); while ( t.isPlaying( ) );
  t.play( NOTE_D4, WHOLE/16 );   while ( t.isPlaying( ) );
  t.play( NOTE_E4, QUARTER );    while ( t.isPlaying( ) );
  t.play( NOTE_D4, QUARTER  );   while ( t.isPlaying( ) );
  t.play( NOTE_G4, QUARTER  );   while ( t.isPlaying( ) );
  t.play( NOTE_FS4, HALF );      while ( t.isPlaying( ) );
}

void loop( ) { }
```





Coding: “D48.D46 E44 D44 G44 FS42



```
#include <Tone.h>
const int WHOLE = 1248, HALF=WHOLE/2, QUARTER=WHOLE/4; // times in ms

void setup( ) {
  Tone t;          // Tone-object
  t.begin( 8 ); // tonePin = 8
  t.play( NOTE_D4, WHOLE*3/16 ); while ( t.isPlaying( ) );
  t.play( NOTE_D4, WHOLE/16 );   while ( t.isPlaying( ) );
  t.play( NOTE_E4, QUARTER );    while ( t.isPlaying( ) );
  t.play( NOTE_D4, QUARTER  );   while ( t.isPlaying( ) );
  t.play( NOTE_G4, QUARTER  );   while ( t.isPlaying( ) );
  t.play( NOTE_FS4, HALF );      while ( t.isPlaying( ) );
}

void loop( ) { }
```

Crying for a **String** of note-names

Done compiling.

```
Sketch uses 2766 bytes (8%) of program storage space. Maximum is 32256 bytes.
Global variables use 31 bytes (1%) of dynamic memory, leaving 2017 bytes for local variables.
```

2 Arduino/Genuino Uno on /dev/ttyACM0

A simple **strategie of playing melodies** using

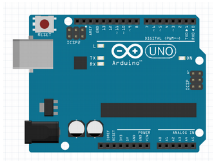
- note-Strings and
- the original Tone-library with only
  - `void **play**( frequency, duration ) and`
  - `bool **isPlaying**( )`

Preparation:
set a pointer to the beginning of the note-string.

Playing:

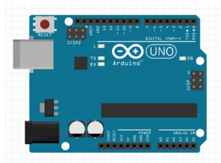
```
while ( not_at_the_end_of_the_note-string ) {
    get_next_note( );
    start_playing_it_for_its_duration( );
    while ( it_is_playing( ) ) ; // Active wait / busy wait.    }
```

Do nothing !
Wasted Time !
No time for anything else !



My **TonePlus**-Library extends the **Tone**-Library

```
class Tone {
  public:
    void begin(uint8_t tonePin);
    bool isPlaying();
    void play(uint16_t frequency, uint32_t duration = 0 );
    void stop();
    void pause( uint32_t duration );
    void delay( uint32_t duration );
  private:
    static int _tone_pin_count;
    uint8_t _pin;
    int8_t  _timer;
};
```



First extension of the class Tone

1) Brett Hagmans Tone-library had no **`pause`**-function.

My first attempt:

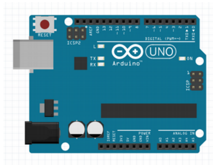
```
void pause( uint32_t duration )
{ play( 24000,duration ); }
```

Negative:
- Some pupils still heard this frequency.
- 24 kHz needs a lot of timer interrupts !

The ideas behind the realized version are
- **not toggling the output** (affords few modifications)
- **use the frequency 0 Hz** as pause indication (internally 8 kHz).

So in **TonePlus**:

```
void pause( uint32_t duration )
{ play( 0,duration ); }
```

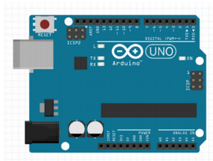


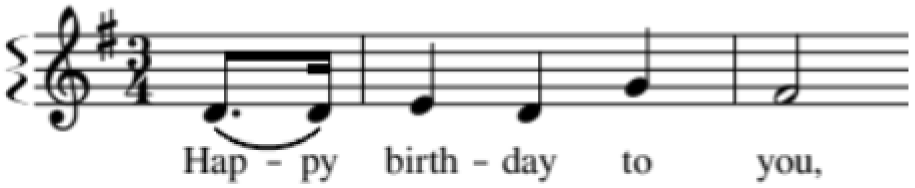
Second extensions of the class Tone

2) Brett Hagmans Tone-library had no **delay**-function.

The realized version in **TonePlus** is

```
void delay( uint32_t duration )
{ pause( duration ); while ( isPlaying( ) ); }
```





Coding: "D48.D46 E44 D44 G44 FS42

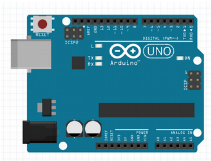
With my TonePlus-library

```
#include <Voices.h>
const char[ ] PROGMEM HAPPY = "D48.D46 E44 D44 G44 FS42"
"D48.D46 E44 D44 A44 G42"
"D48.D46 D54 H44 G44 FS44 E44"
"C58.C56 H44 G44 A44 G42.";

void setup( ) {
  Voice v1( 8, HAPPY );
  v1.play( );
  v1.delay( 5000 );    // 5 s pause ( active wait )
}

void loop( ) { }
```

PROGMEM: The constant char-Array is stored in the bigger program memory.

With 5% **pauses** at the end of not connected notes.

The global delay-function is not available, when all timers are used.



III. Playing multi-sound melodies

Under https://create.arduino.cc/projecthub/liss
I once found a small project for playing “Happy Birthday”
with 3 sounds (Tone-objects solo, bass, rhythm) and 3 LEDs.

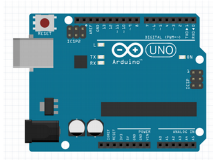
```
void loop(void) {
 bass.play (NOTE_G3, t);    switchBassLed();
  rythm.play(NOTE_G4, t24); switchRythmLed();
   solo.play(NOTE_E4, t);   switchSoloLed();
    wait(rythm);
  rythm.play(NOTE_B4, t14); switchRythmLed();
    wait(rythm);
  rythm.play(NOTE_D5, t14); switchRythmLed();
    wait(rythm);
. . .

and further 190 lines
```

durations

1/1 s
2/4 s
1/1 s

1/4 s

1/4 s

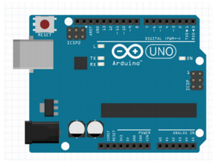


A better idea would be to

- **use note-strings** (solo, bass and rhythm)

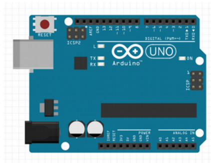
- **use a scheduler to manage the sequencing of the different voices correctly:**

  while (isPlaying( )) for the next ending tone;
  start the next tone for this just ended voice;



- **Write the note-strings** (solo, bass and rhythm).

```
const char PROGMEM SOLO[ ] =
  "D48.D46 E44 D44 G44  FS42 D48.D46 " // 1., 2.
  "E44 D44 A44     G44       D48.D46 " // 3., 4.
  "D54 H44 G44     FS44 E44  C58.C56 " // 5., 6.
  "H44 G44 A44     G42.";              // 7., 8.
const char PROGMEM BASS[ ] =
  "P4 G34 H34 D44  D44 FS44 A44 "      // 1., 2.
  "D44 FS44 A44    G34 H34  D44 "      // 3., 4.
  "G34 H34 D44     G44 E44  G44 "      // 5., 6.
  "G34 D44 FS34    G34 H34  H34  ";    // 7., 8.
const char PROGMEM RHYTHM[ ] =
  "P4  G48  H46 D56 G48  H46 D56 G48  H46 D56 "// 1.
  "D58 FS56 A56 D58 FS56 A56 D58 FS56 A56 "    // 2.
  "D58 FS56 A56 D58 FS56 A56 D58 FS56 A56 "    // 3.
  "G48 H46  D56 G48 H46  D56 G48 H46  D56 "    // 4.
  "D58 FS56 A56 D58 FS56 A56 D58 FS56 A56 "    // 5.
  "C58 E56  G56 C58 E56  G56 C58 E56  G56 "    // 6.
  "G48 H46  D56 G48 H46  D56 D58 FS56 A56 "    // 7.
  "G48 H46  D56 G48 H46  D56 P4 ";             // 8.
```



Realize / program the idea

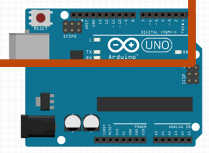
Preparation

- For all voices v0, v1, v2:
  set their startpointers (to beginning of their note-string).

Playing multisound melodies:

- Start the **Scheduler**

```
while ( not_at_the_end_of_all_note_strings ) {
  v = tone_with_the_shortest_remaining_duration( );
  d = remaining_duration( v );
  reduce_the_remaining_durations_of_all_voices_by( d );
  while ( v.isPlaying( ) ); // active ( busy ) waiting !!
  v.get_and_play_next_tone( );
}
```

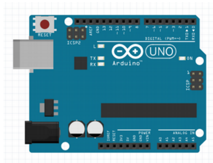


Concept for the new library Voices

Contains only one class **Voice** with

- attributes for **notes**
  - frequency, duration, …
  - pointers to note-strings
    - startPtr, ptr
  - **static array with pointers to Voice-objects**
- all **note-strings are stored in program-memory** to save precious RAM place.
- additional syntactic elements for the note-string:

|: :|	for repetition of parts of the note-string
!	as information for the first voice of a **canon** to start the next voice at the beginning.



IV. Playing canons

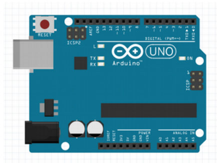
Strategy:

All voices get the same melody.

Only the first voice starts playing.

When the first voice reaches an ‘!’ in the note-string,
- start the next voice (this one has to ignore ‘!’s)

Stop playing, when the first voice reaches
the end of the note-string for the second time.

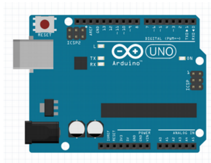


V. Used features of C++

This project “Playing Multisound Melodies” with
the aim of a Multisound Library is highly motivating.

It affords experimental programming using the Arduino
and leads to experimental learning many C++ features:

- classes with attributes and methods (Voice, play, ...)
- static functions and variables (static Voice* arrays, …)
- templates (functions with different numbers of parameters)
- scheduling (playing multiple voices)
- interrupts (as alternative to polling / scheduling)
- multitasking (possible when avoiding active waiting)



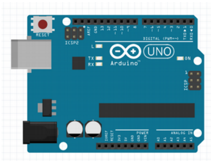
Using Interrupts is a much better alternative

Scheduler

```
while ( not_at_the_end_of_all_note_strings ) {
  v = tone_with_the_shortest_remaining_duration( );
  d = remaining_duration( v );
  reduce_the_remaining_durations_of_all_voices_by( d );
  while ( v.isPlaying( ) );
  v.get_and_play_next_tone( );
}
```

InterruptHandler: ISR: Interrupt Service Routine

```
void ISR( ) {    // Called when a tone ends.
  v = tone_that just ended;
  v.get_and_play_next_tone( );
}
```



Using Interrupts is a much better alternative

InterruptHandler: ISR: Interrupt Service Routine

```
void ISR( ) {  // Called when a tone ends.
  v = tone_that just ended;
  v.get_and_play_next_tone( );
}
```

BUT:

ISRs should be as short as possible !

ISRs should be statically !

May interrupts happen during interrupt handling ?

What is with interrupt queuing ?

All is manageable !!!

Perfect teaching material for C++