

## Opgave 12 Spilledåse opgave

Opgaven går ud på at lave et program der kan spille en melodi efter noder. Du kan køre programmet på en Arduino. Det er lettest at starte på en PC udvikling af prgrammet på pc – benyt arduino plugin på visual studio og på visual code studio.

Noderne kodes som **en tekststreng**, hvor et **bogstav angiver tonen** og et **efterfølgende tal** angiver **længden**. Programmet skal fortolke tekststrengen og samtidigt spille melodien i en højttaler/buzzer tilkoblet arduino.

Tonerne er angivet således:



Tonen 'b' kaldes også 'h'. 'p' angiver en pause.

Et '#' efter tonen angiver en halv tone højere. Et '-' angiver en halv tone lavere. For eksempel kan en halv tone højere end F (Fis) angives som 'f#' eller 'g-'. Et '>' angiver at tonen er en oktav højere. '<' angiver en oktav lavere.






Tonehøjden i forhold til kammertonen (A) kan måles i halvtoner. Der er kun en halv tone mellem E og F og mellem H og C, mens der er en hel tone mellem de øvrige toner.

Tone	Kode	Halvtoner over kammertonen	Frekvens, Hz
H	h<	-10	247
C	c	-9	262
Cis	c#, d-	-8	277
D	d	-7	294
Dis	d#, e-	-6	311
E	e	-5	330
F	f	-4	349
Fis	f#, g-	-3	370
G	g	-2	392
Gis	g#, a-	-1	415
A	a	0	440
Ais	a#, h-, b-	1	466
H	h, b	2	494
C	c>	3	523

Toneskalaen er logaritmisk, således at frekvensen fordobles for hver oktav. Der er 12 halvtoner per oktav, således at frekvensen ganges med  $2^{1/12}$  for hver halvtone vi går op, og divideres med  $2^{1/12}$  for hver halvtone vi går ned. Kammertonen (A) er per definition 440 Hz, og de øvrige toner beregnes ud fra denne. Det er ikke

nogen god ide at beregne frekvenserne i en Arduino, da den ikke har indbygget *floating point processor*. I stedet skal programmet indeholde en tabel over frekvenserne, afrundet til heltal.

Tonens længde er angivet således:

	= 1	1/16
	= 2	1/8
	= 4	1/4
	= 8	1/2
	= 16	1

Tempoet er defineret ved hjælp af en tidsenhed der angives med et 't' efterfulgt af et tal i millisekunder.

Således, at en helnode er 128 tidsenheder og en sekstendedels node er **8 tidsenheder**. Og en fjerdedels node er 4 gange længere i tid, så nedenstående eksempel er tidsenheden 10 ms (t10), således at f.eks. "f4"

♩ => 8\*4\*10 ms = 320 ms, inkl. pause 1/8. Heri er inkluderet en pause, som er 1/8 af den samlede tid.

Tonen J (der angives som 4 i koden) lyder altså i 7\*4\*10 ms = 280 ms, efterfulgt af en pause på 1\*4\*10 ms = 40 ms. En tilsvarende pause (p4 i koden) varer 8\*4\*10 ms = 320 ms.

Her er et eksempel der viser hvordan noderne til "Mester Jakob" er omsat til denne kode.

**Mester Jakob**

Musik: Anonym



Tekst: Anonym

Kodestreg for Mester Jakob:

"t10,f4,g4,a4,f4,f4,g4,a4,f4,a4,h-4,c>8,a4,h-4,c>8,c>2,d>2,c>2,h-2,a4,f4,c>2,d>2,c>2,h-2,a4,f4,f4,c4,f8,f4,c4,f8,"

t10 er tidsfaktor der skal ganges på alle toner

Nedenfor er nogle flere eksempler på melodier du kan bruge.

Programmet skal bruge en [tilstandsmaskine](#) til at fortolke kodestrengen. Tilstandsmaskinen skal læse et tegn ad gangen hvorefter det skifter tilstand afhængigt af det pågældende tegn. I start tilstanden forventer vi at læse en tone 'a' til 'h' eller 'p' eller 't'. Efter, for eksempel, at have læst tegnet 'a' er vi i en tilstand, hvor vi forventer at det næste tegn er et ciffer '0' – '9' eller et af tegnene '#', '-', '>', '<'. Efter, for eksempel, at have læst et ciffer er vi nu i en tilstand hvor vi forventer enten endnu et ciffer eller et komma. Efter et komma spiller vi den pågældende tone, hvorefter vi går tilbage til start-tilstanden og fortsætter. Hvis der er en syntaksfejl, skal programmet stoppe.

Tegn et [tilstandsdiagram](#) før du begynder at kode tilstandsmaskinen. Nedenfor er et forslag til tilstandsdiagram.

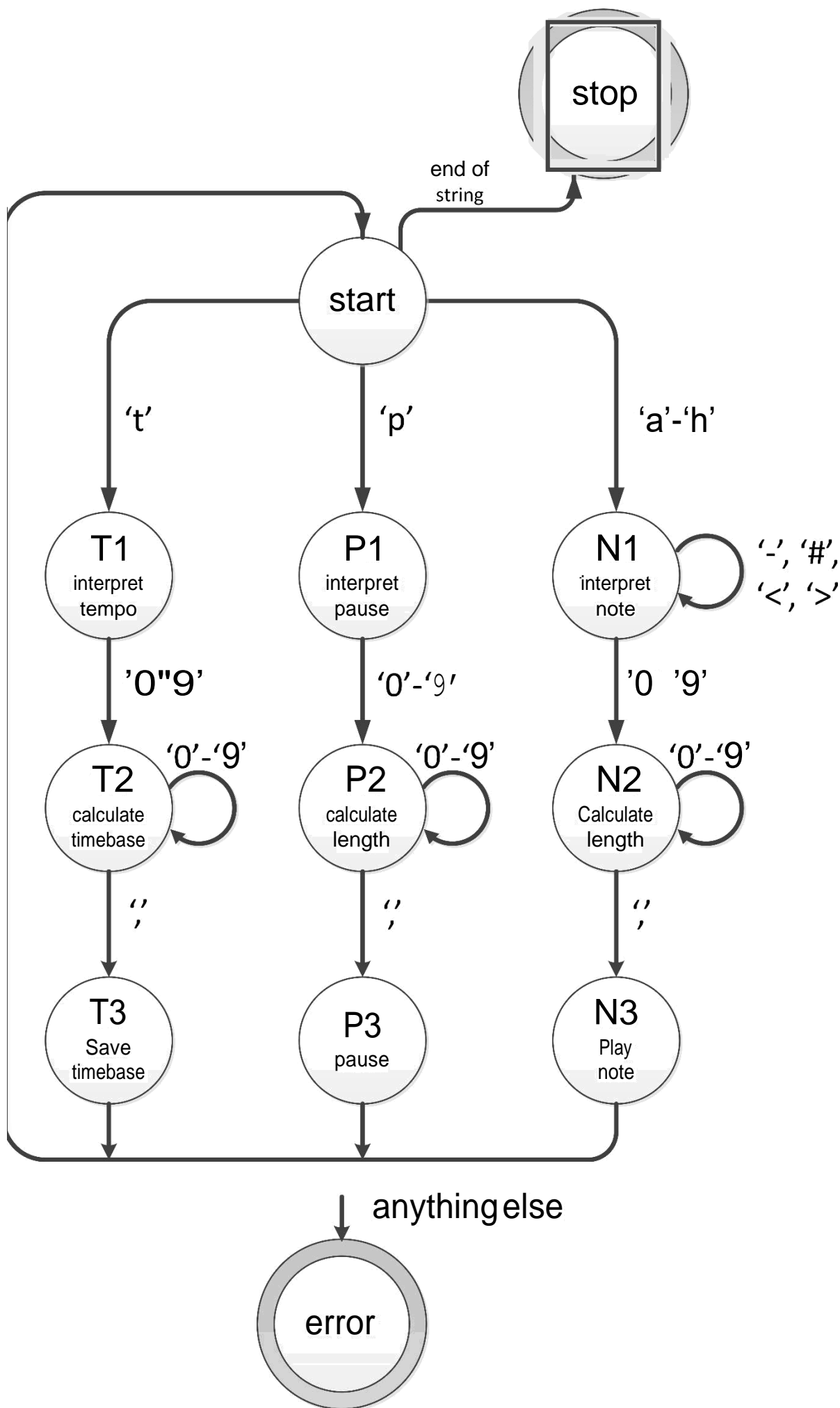
Det er lettest at lave programmet i Windows først fordi du har adgang til en debugger der kan køre én linje kode ad gangen. Til gengæld er lyden bedre på en Arduino. Når du har fået programmet til at virke i Windows kan du overføre det til Arduino.

I Windows kan du lave en tone med funktionen Beep(frequency, duration), og en pause med funktionen Sleep(duration).

På Arduino kan du lave en tone med funktionen tone(pin, frequency, duration), og en pause med funktionen delay(duration).

## PS.

Vær hensynsfuld. Lad være med at forstyrre dine medstuderende og andre mere end nødvendigt når du afprøver dit program. Skru ned for lyden eller brug hovedtelefoner.



Her ses et eksempel på tilstandsdiagram. Nedenfor er **et skelet** til spilledåse-programmet.

```
/*
| spilledåse.cpp
| Version: 2.00
| Created by: Agner Fog
| Created date: 2018-11-20
| Last modified: 2019-11-19
| Description: Example of music box playing a tune after notes sheet.
*/

#include <stdio.h>
#include <windows.h>

// Code string for tune:
// Mester Jakob
const char jakob[] = "t10,f4,g4,a4,f4,f4,g4,a4,f4,a4,h-4,c>8,a4,h-4,c>8,"
                    "c>2,d>2,c>2,h-2,a4,f4,c>2,d>2,c>2,h-2,a4,f4,f4,c4,f8,f4,c4,f8,";

// table of frequencies
const unsigned int freq[36] = {
    // c      d      e      f      g      a      h
    131, 139, 147, 156, 165, 175, 185, 196, 208, 220, 233, 247, // low octave
    262, 277, 294, 311, 330, 349, 370, 392, 415, 440, 466, 494, // default octave
    523, 554, 587, 622, 659, 698, 740, 784, 831, 880, 932, 988 // high octave
};

// Name states for state machine
enum States
{
    Start, // start or after comma
    T1,    // after 't': interpret timebase
    T2,    // after T1 and at least one digit: calculate tempo
    T3,    // after T2 and comma: save tempo
    N1,    // after note a-h or #-<>: Interpret note
    N2,    // after N1 and at least one digit: Calculate duration
    N3,    // after N2 and comma: Play note
    P1,    // after 'p'. Interpret pause
    P2,    // after P1 and at least one digit: Calculate duration
    P3,    // after P2 and comma: Play pause
    Stop,  // finished
    Error  // syntax error
};

// The class StateMachine defines a state machine for interpreting a music string
class StateMachine
{
public:
    void play(const char *tune); // play the tune protected:
    void SStart();              // start state
    void ST1();                  // state T1: interpret tempo
    void ST2();                  // state T2: calculate tempo
    void ST3();                  // state T3: save tempo
    void SP1();                  // state P1: interpret pause
    void SP2();                  // state P2: calculate length of pause
    void SP3();                  // state P3: play pause
    void SN1();                  // state N1: interpret note
    void SN2();                  // state N2: calculate length of note
    void SN3();                  // state N3: play note
    void SError();               // state error
    States state;                // state in syntax parsing

    int timebase; // time unit int pitch; // note to play
    int duration; // duration of note or pause
    const char * p; // position in tune string
    char c; // current character in tune string
};

// function to play a tune
void StateMachine::play(const char *tune)
```

```

{
    p = tune; // pointer to tune string timebase = 20;    // default time base
    state = Start;    // start in state Start
    c = *p;           // read first character
    p++;              // advance pointer for reading next character

    // run state machine
    while (state != Stop)
    {
        switch (state)
        {
            case Start: // Read next item SStart();        break;

            case T1: // interpret timebase ST1();    break;

            case T2: // state T2: calculate timebase ST2();    break;

            case T3: // state T3: save timebase ST3();    break;

            case P1: // state P1: interpret pause SP1(); break;

            case P2: // state P2: calculate length of pause SP2();    break;

            case P3: // state P3: play pause SP3();    break;

            case N1: // state N1: interpret note SN1();    break;

            case N2: // state N2: calculate length of note SN2();    break;

            case N3: // state N3: play note SN3(); break;

            case Error: // state error SError();
            default:
                break;

        } // end switch (state)
    } // end while
}

// Functions for each state:

void StateMachine::SStart()
{ // start state
    switch (c)
    {
        case 't': // interpret timebase
            state = T1;
            break;
        case 'p': // pause
            state = P1;
            break;
        case 'a': // note a - h
        case 'b':
        case 'c':
        case 'd':
        case 'e':
        case 'f':
        case 'g':
        case 'h':
            state = N1;
            break;
        case ',': // ignore comma break;
        case 0: // end of string
            state = Stop;
            break;
        default: // anything else should give an error state = Error;
            break;
    }
}

/* Next comes the functions for each state.
Each function should do the following:

```

1. Interpret the character c, and do any necessary calculations
2. Read the next character and advance the pointer p
3. Set the next state, depending on the new character

```

*/

//Her er flere melodier :

// Pippi Langstrømpe
const char pippi[] = "t20,c2,f1,f1,a2,f2,g4,h-1,a1,g1,f1,e2,g1,e1,c2,e2,f4,a4, "
    "c2,f2,a2,f2,g4,h-1,a1,g1,f1,e2,g1,e1,c2,e2,f6,p2,c2,f2,a2,f2,g4,h-1,a1,g1,f1, "
    "e2,g1,e1,c2,e2,f4,a4,c2,f2,a2,f2,g4,h-1,a1,g1,f1,e2,g1,e1,c2,e2,f6,p2,a2,a1, "
    "a1,a2,a2,h-3,h-3,h-1,a1,g2,g1,g1,g2,g1,f1,e2,f2,g2,p2,a2,a1,a1,a2,a2,h-4,h-2, "
    "h-1,a1,g2,g2,f2,e2,f8,";

// Åh abe
const char abe[] = "t10,g2,c>4,g4,h4,f4,g4,e4,p4,f4,g4,c>4,h4,a4, "
    "g8,p3,e1,f3,f#1,g3,a1,g3,e1,f3,g1,f3,d1,e4,c>4,h4,a4,g4,h3,a1,g4,h3,a1, "
    "g3,f1,e3,d1,c4,p4,f8,d4,c4,f8,d4,c4,f8,d4,c4,f4,f4,f4,p4,g8,e4,d4, "
    "g8,e4,d4,g4,h4,a4,e4,g4,g4,p2,";

// Olsen Banden
const char olsen[] = "t8,d4,e-4,d2,e-4,f10,p8,d4,e-4,f2,g4,a-10,p8,g4,a-4,g4, "
    "a-2,h-4,h-4,h-2,a4,g4,f4,f4,f-4,e-4,d4,f2,f4,g2,f4,e-16,p4,e-2,e-4,f2,e-4, "
    "d16,p4,f2,f4,g2,f4,e-16,p4,e-2,e-4,f2,e-4,d32,c>4,c#>4,d>4,e>4,f>4,c>4,p8, "
    "p4,c>2,c>4,c->2,h-4,a16,c>4,c#>4,d>4,e>4,f>4,c>4,p8,p4,c>2,c>4,d>2,e>4,f>16, "
    "d4,e-4,d2,e-4,f10,p8,d4,e-4,f2,g4,a-10,p8,g4,a-4,g4,a-2,h-4,h-4,h-2,a4,g4,f4, "
    "f4,f-4,e-4,d4,f2,f4,g2,f4,e-16,p4,e-2,e-4,f2,e-4,d16,p4,f2,f4,g2,f4,e-16,p4, "
    "f2,f4,g2,a4,h-4,p4,h-<4,p4,";

// Marseillaisen
const char marseille[] = "t20,d1,d3,d1,g4,g4,a4,a4,d>6,h2,g2,p1,g1,h3,g1, "
    "e4,c>8,a3,f#1,g8,p3,g1,g3,a1,h4,h4,h4,c>3,h1,h4,a4,p4,a3,h1,c>4,c>4,c>4,d>3, "
    "c>1,h8,p4,d>3,d>1,d>4,h3,g1,d>4,h3,g1,d8,p3,d1,d3,f#1,a8,c>4,a3,f#1, "
    "a4,g4,f8,e4,g3,g1,g4,f#3,g1,a8,p4,p2,a2,h-6,h-2,h-2,h-2,c>2,d>2,a12,h-2,a2, "
    "g6,g2,g2,h-2,a2,g2,g4,f#2,p8,p1,d>1,d>11,d>1,h3,g1,a12,p3,d>1,d>11,d>1,h3,g1, "
    "a10,p2,d4,g8,p4,a4,h8,p8,c>8,d>4,e>4,a10,p2,e>4,d>11,h1,c>3,a1,g12,p4,";

// Sound of silence
const char silence[] = "t30,p2,d1,d1,f1,f1,a1,a1,g8,p1,c1,c1,c1,e1,e1,g1,g1, "
    "f8,p1,f1,f1,f1,a1,a1,c>1,c>1,d>4,c>4,p2,f1,f1,a1,a1,c>1,c>1,d>4,c>4,p2,f1,f1, "
    "d>1,d>5,d>1,e>1,f>1,f>3,e>1,d>3,c>6,d>1,c>1,a8,p1,f1,f1,f1,c>6,p1,e1,f1,d7,";

int main()
{
}

```