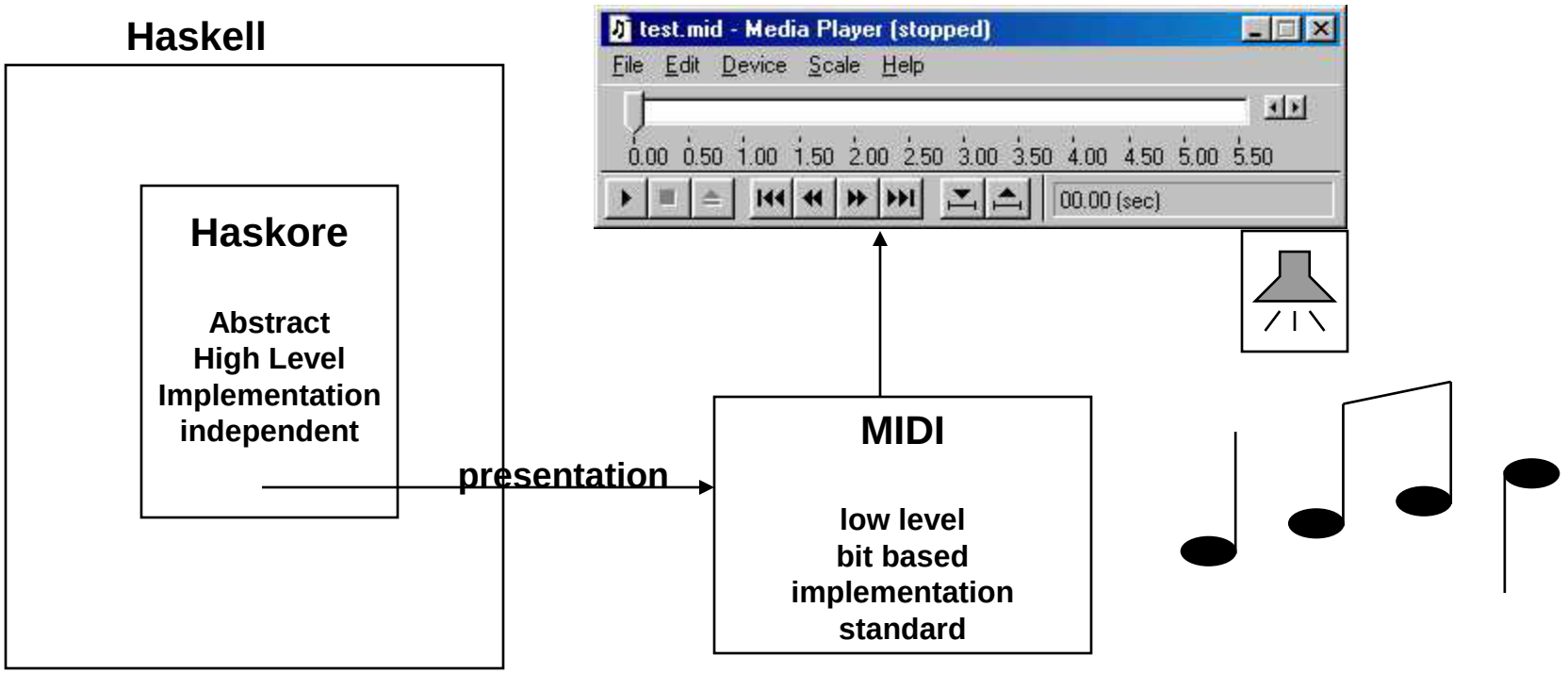
Compositional Functional Programming with the Haskore Music

- **Todays Topics**
  - **The Haskore System**
  - **The `Music` datatype**
  - **MIDI Instruments**
  - **Pitch & absolute Pitch**
  - **Composing Music**
    - » **Delay**
    - » **Repeating**
    - » **Transposing**
  - **Manipulating Music**
    - » **Duration**
    - » **Cutting**
    - » **Reversing**
  - **Percussion**
  - **Presentation and the MIDI file format**

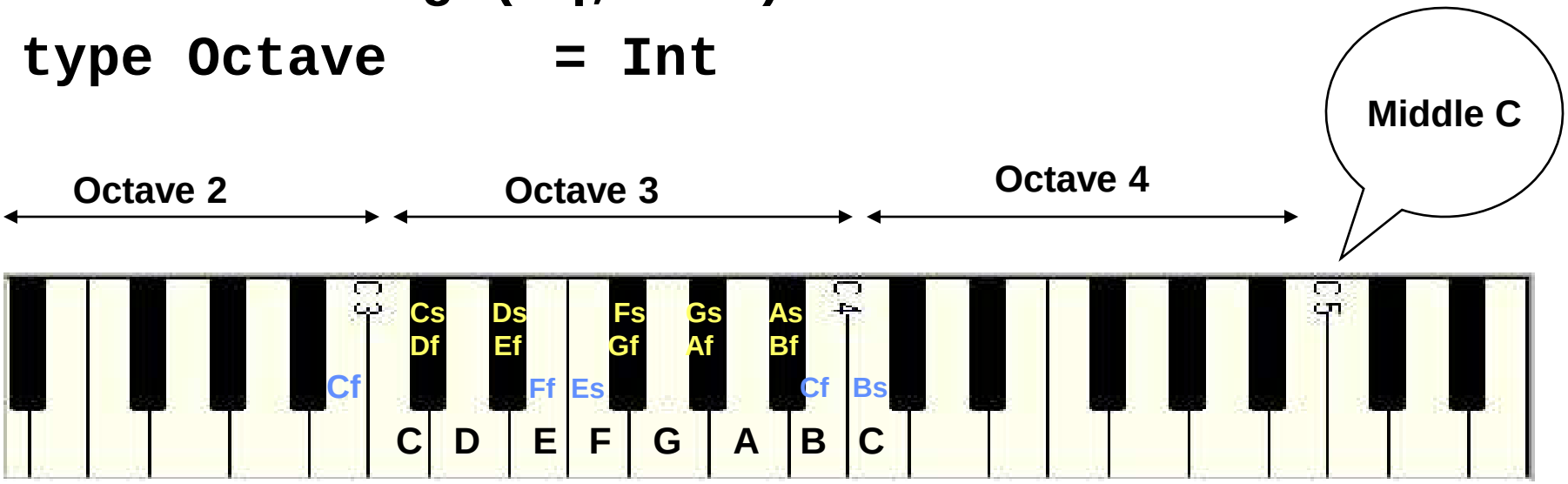
Haskore

- **Haskore is a Haskell library for constructing digital music**
  - **It supports an abstract high-level description of musical concepts**
  - **Maps into the Midi (Musical Instrument Digital Interface) standard**
    - » **a low-level binary bit based encoding of music**
    - » **can be "played" by "Media-Players"**



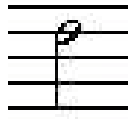
Musical Basics in Haskore

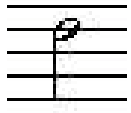
```
type Pitch      = (PitchClass, Octave)
data PitchClass =
    Cf | C  | Cs | Df | D  | Ds | Ef | E
  | Es | Ff | F  | Fs | Gf | G  | Gs | Af
  | A  | As | Bf | B | Bs
      deriving (Eq,Show)
type Octave     = Int
```

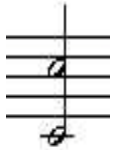
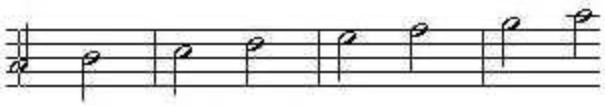


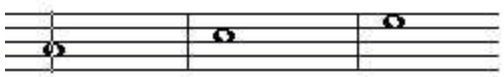
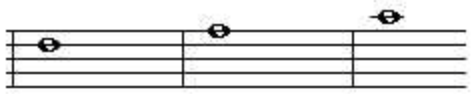
Music

```
data Music = Note Pitch Dur
    | Rest Dur
    | Music :+: Music
    | Music :=: Music
    | Tempo
        (Ratio Int) Music
    | Trans
        Int Music
    | Instr  IName Music
```






Midi Standard supports lots of instruments

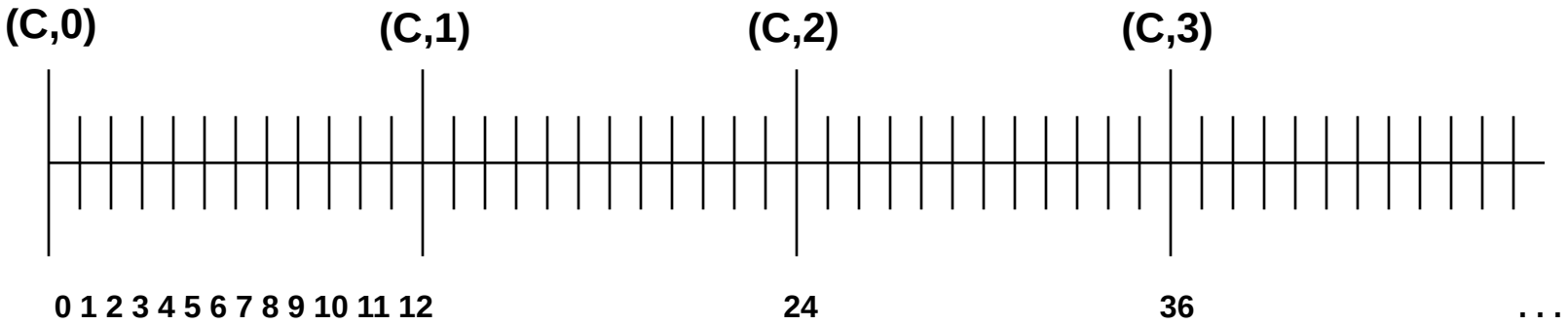
```
data IName
 = AcousticGrandPiano    | BrightAcousticPiano | ElectricGrandPiano  | HonkyTonkPiano  |
   RhodesPiano         | ChorusedPiano       | Harpsichord         | Clavinet
   | Celesta            | Glockenspiel        | MusicBox            | Vibraphone
   | Marimba            | Xylophone           | TubularBells        | Dulcimer
   | HammondOrgan       | PercussiveOrgan     | RockOrgan           | ChurchOrgan
   | ReedOrgan          | Accordion           | Harmonica           | TangoAccordion  |
   AcousticGuitarNylon | AcousticGuitarSteel | ElectricGuitarJazz  | ElectricGuitarClean
   | ElectricGuitarMuted | OverdrivenGuitar  | DistortionGuitar      | GuitarHarmonics
   | AcousticBass  | ElectricBassFingered  | ElectricBassPicked  | FretlessBass  |
   SlapBass1           | SlapBass2           | SynthBass1      | SynthBass2  | Violin
   | Viola             | Cello          | Contrabass  | TremoloStrings        |
   PizzicatoStrings    | OrchestralHarp  | Timpani  | StringEnsemble1       |
   StringEnsemble2     | SynthStrings1 | SynthStrings2         | ChoirAahs           |
   VoiceOohs       | SynthVoice  | OrchestraHit          | Trumpet             |
   Trombone        | Tuba  | MutedTrumpet          | FrenchHorn          | BrassSection
   | SynthBrass1  | SynthBrass2           | SopranoSax          | AltoSax         |
   TenorSax  | BaritoneSax           | Oboe   | Bassoon          | EnglishHorn       |
   Clarinet  | Piccolo               | Flute              | Recorder | PanFlute  |
   BlownBottle  | Shakuhachi           | Whistle           | Ocarina         |
   Lead1Square  | Lead2Sawtooth        | Lead3Calliope     | Lead4Chiff  |
   Lead5Charang    | Lead6Voice         | Lead7Fifths  | Lead8BassLead         |
   Pad1NewAge          | Pad2Warm  | Pad3Polysynth         | Pad4Choir           |
   Pad5Bowed  | Pad6Metallic          | Pad7Halo          | Pad8Sweep  | FX1Train
   | FX2Soundtrack       | FX3Crystal  | FX4Atmosphere         | FX5Brightness       |
   FX6Goblins  | FX7Echoes             | FX8SciFi          | Sitar           | Banjo
   | Shamisen  | Koto                  | Kalimba           | Bagpipe         | Fiddle
   | Shanai  | TinkleBell            | Agogo             | SteelDrums      | Woodblock
   | TaikoDrum  | MelodicDrum          | SynthDrum         | ReverseCymbal  |
   GuitarFretNoise       | BreathNoise         | Seashore  | BirdTweet           |
   TelephoneRing       | Helicopter  | Applause    | Gunshot  | Percussion
          deriving (Show,Eq,Ord,Enum)
```

Duration & Absolute Pitch

```
type Dur   = Ratio Int
```

- fractions of Integers such as 3 /4. We write `(3 % 4)` in Haskell.

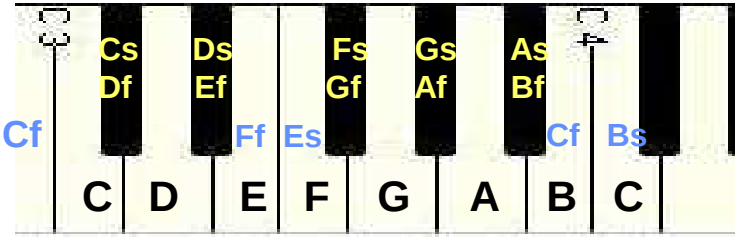
```
type AbsPitch = Int

absPitch :: Pitch -> AbsPitch
absPitch (pc,oct) = 12*oct + pcToInt pc
```



Pitch to integer

```
pcToInt :: PitchClass -> Int
pcToInt pc = case pc of
               Cf -> -1   -- should Cf be 11?
               C  -> 0  ;  Cs -> 1
               Df -> 1  ;  D  -> 2 ;  Ds -> 3
               Ef -> 3  ;  E  -> 4 ;  Es -> 5
               Ff -> 4  ;  F  -> 5 ;  Fs -> 6
               Gf -> 6  ;  G  -> 7 ;  Gs -> 8
               Af -> 8  ;  A  -> 9 ;  As -> 10
               Bf -> 10 ;  B  -> 11 ; Bs -> 12 -- maybe 0?
```

Note how several different pitches have the same absolute pitch. This is because the “flat” of some notes is the “sharp” of another.

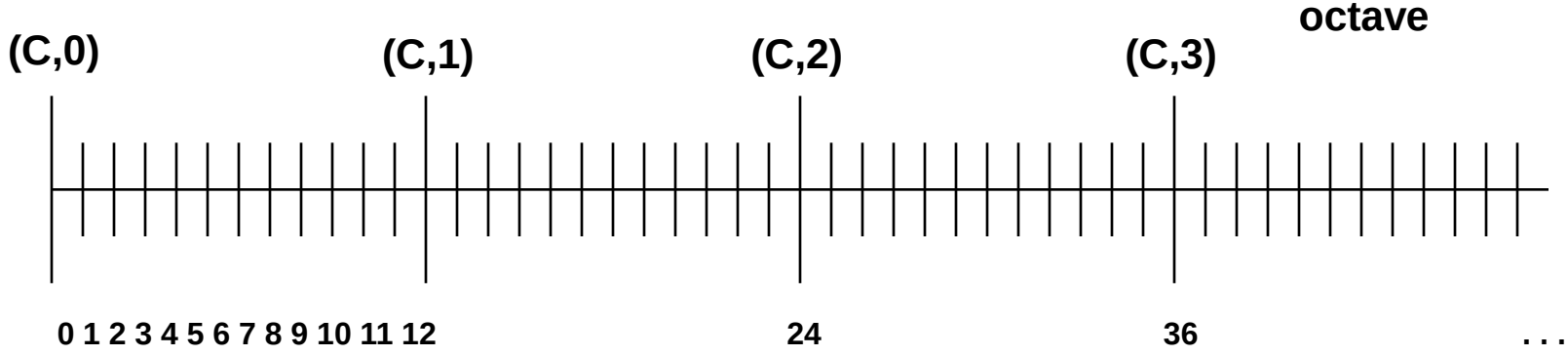


From AbsPitch to Pitch

```
pitch12 = [C,Cs,D,Ds,E,F,Fs,G,Gs,A,As,B]
pitch    :: AbsPitch -> Pitch
pitch a = (pitch12 !! mod a 12, quot a 12)
```

Dist above C

octave




```
trans    :: Int -> Pitch -> Pitch
trans i p = pitch (absPitch p + i)
```

Generic Music - Notes

```
cf,c,cs,df,d,ds,ef,e,es,ff,f,fs,gf,g,gs,af,a,as,bf,b,bs
  :: Octave -> Dur -> [NoteAttribute] -> Music

cf o = Note(Cf,o);  c o = Note(C,o);  cs o = Note(Cs,o)
df o = Note(Df,o);  d o = Note(D,o);  ds o = Note(Ds,o)
ef o = Note(Ef,o);  e o = Note(E,o);  es o = Note(Es,o)
ff o = Note(Ff,o);  f o = Note(F,o);  fs o = Note(Fs,o)
gf o = Note(Gf,o);  g o = Note(G,o);  gs o = Note(Gs,o)
af o = Note(Af,o);  a o = Note(A,o);  as o = Note(As,o)
bf o = Note(Bf,o);  b o = Note(B,o);  bs o = Note(Bs,o)
```

Given an **`Octave`** creates a function from **`Dur`** to **`Music`** in that octave.
Note that **`Note :: Pitch -> Dur -> Music`**

These functions have the same names as the constructors of the **`PitchClass`** but they're not capitalized.

Generic Music - Rests

```
wn,  hn,  qn,  en,  sn,  tn  :: Dur
dhn, dqn, den, dsn           :: Dur
wnr, hnr, qnr, enr, snr, tnr :: Music
dhnr, dqnr, denr, dsnr       :: Music

wn  = 1     ; wnr  = Rest wn      -- whole
hn  = 1%2   ; hnr  = Rest hn      -- half
qn  = 1%4   ; qnr  = Rest qn      -- quarter
en  = 1%8   ; enr  = Rest en      -- eight
sn  = 1%16  ; snr  = Rest sn      -- sixteenth
tn  = 1%32  ; tnr  = Rest tn      -- thirty-second

dhn = 3%4   ; dhnr = Rest dhn     -- dotted half
dqn = 3%8   ; dqnr = Rest dqn     -- dotted quarter
den = 3%16  ; denr = Rest den     -- dotted eighth
dsn = 3%32  ; dsnr = Rest dsn     -- dotted sixteenth
```

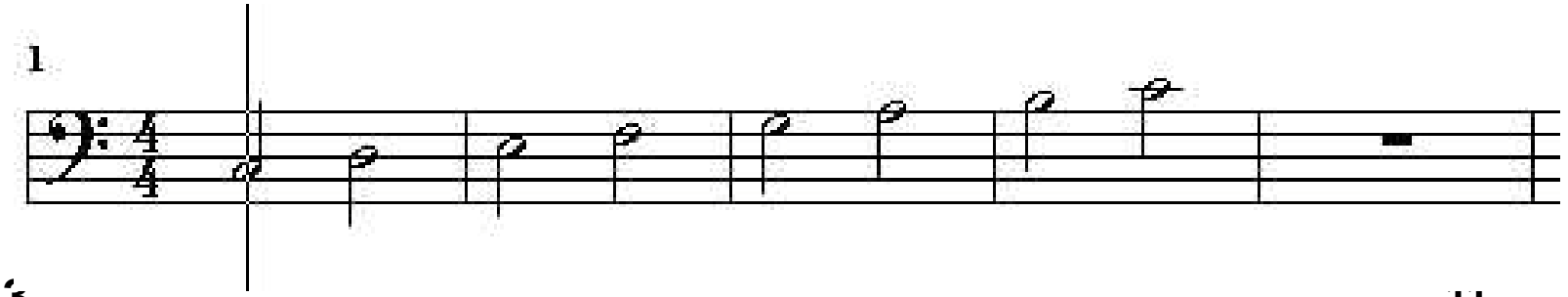
Lets Write Some Music!

```
line, chord :: [Music] -> Music
line  = foldr (:+:) (Rest 0)
chord = foldr (:=:) (Rest 0)
```

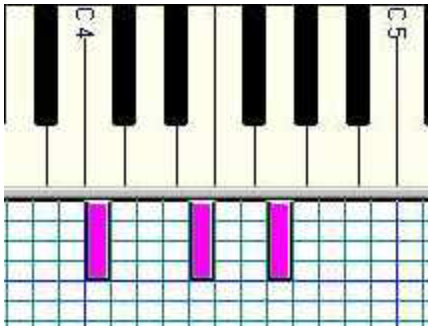
- **Example 1**

Note the change in Octave

```
cScale =
  line [c 4 qn [], d 4 qn [], e 4 qn [],
        f 4 qn [], g 4 qn [], a 4 qn [],
        b 4 qn [], c 5 qn []]
```

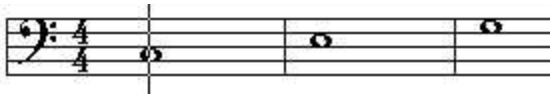


More Examples



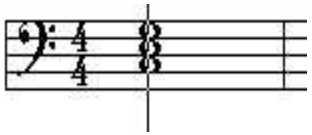

```
cMaj = [ n 4 hn | n <- [c,e,g] ]
cMin = [ n 4 wn | n <- [c,ef, g] ]
```

- **Example 2**



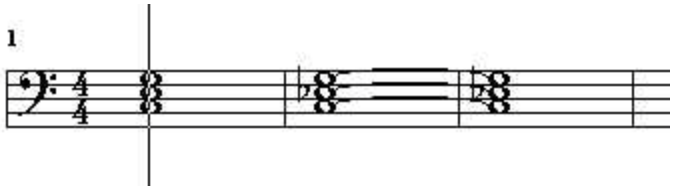
```
cMajArp = line  cMaj
```

- **Example 3**



```
cMajChd = chord cMaj
```

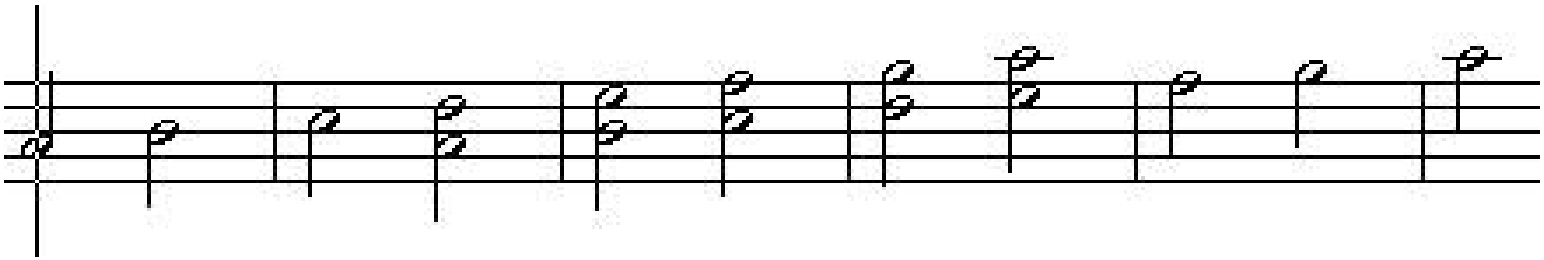
- **Example 4**



```
ex4 = line [ chord cMaj, chord cMin ]
```

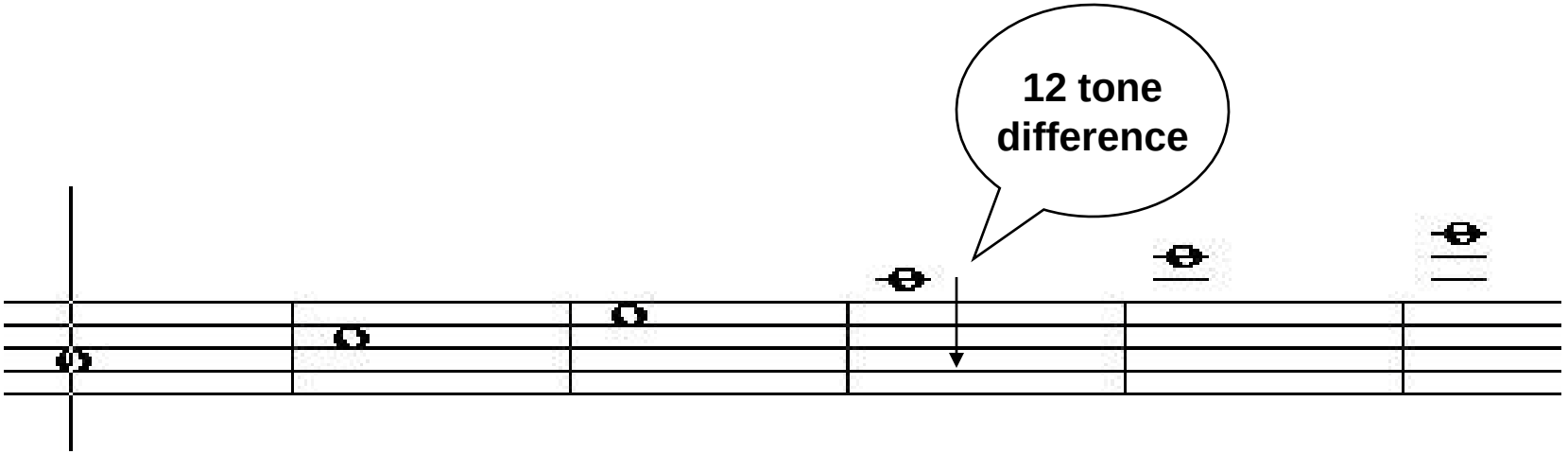
Time Delaying Music

```
delay :: Dur -> Music -> Music
delay d m = Rest d :+: m

ex5 = cScale :=: (delay dhn cScale)
```



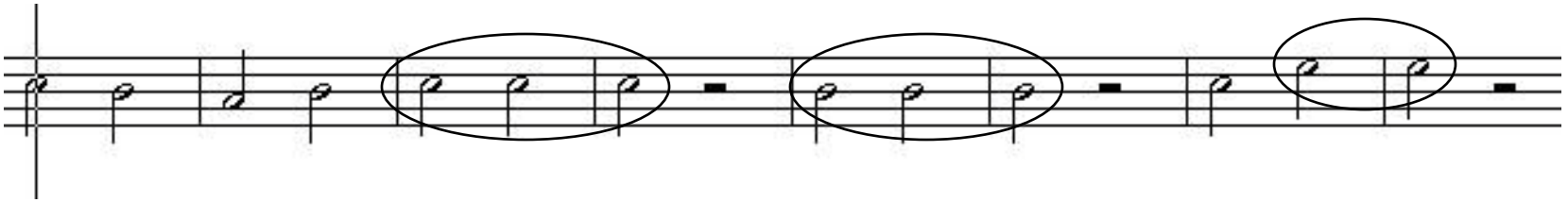
Transposing Music

```
ex6 = line [line cMajor
           ,Trans 12 (line cMajor)]
```



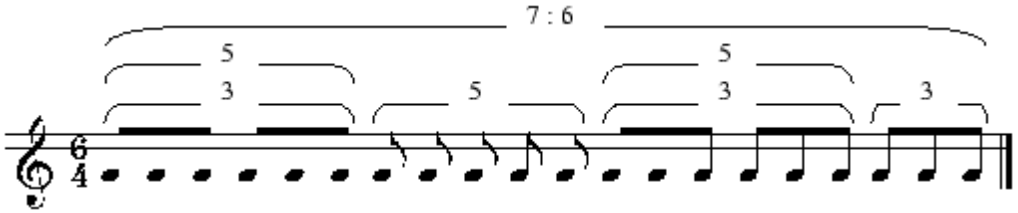
Repeating Music

```
repeatM :: Music -> Music
repeatM m = m :+: repeatM m

nBeatsRest n note =
   line ((take n (repeat note)) ++ [qnr])

ex7 =
  line [e 4 qn [], d 4 qn [], c 4 qn [], d 4 qn [],
        line [ nBeatsRest 3 (n 4 qn []) | n <- [e,d] ],
        e 4 qn [], nBeatsRest 2 (g 4 qn [] ) ]
```



Fancy Stuff

```
pr1, pr2 :: Pitch -> Music
pr1 p = Tempo (5%6)
          (Tempo (4%3) (mkLn 1 p qn :+:
                        Tempo (3%2) (mkLn 3 p en :+:
                                     mkLn 2 p sn :+:
                                     mkLn 1 p qn    ) :+:
                        mkLn 1 p qn) :+:
           Tempo (3%2) (mkLn 6 p en))
pr2 p = Tempo (7%6)
          (m1 :+:
           Tempo (5%4) (mkLn 5 p en) :+:
           m1 :+:
           Tempo (3%2) m2)
  where m1 = Tempo (5%4) (Tempo (3%2) m2 :+: m2)
        m2 = mkLn 3 p en
mkLn n p d = line (take n (repeat (Note p d)))
pr12 :: Music
pr12 = pr1 (C,5) :=: pr2 (G,5)
```



How long is a piece of music?

```
dur :: Music -> Dur

dur (Note _ d)    = d
dur (Rest d)      = d
dur (m1 :+: m2)   = dur m1   +   dur m2
dur (m1 :=: m2)   = dur m1 `max` dur m2
dur (Tempo  a  m) = dur m / a
dur (Trans  _  m) = dur m
dur (Instr  _  m) = dur m
```

Reversing a piece of music

```
revM :: Music -> Music

revM n@(Note _ _) = n
revM r@(Rest _)   = r
revM (Tempo a  m) = Tempo a    (revM m)
revM (Trans i  m) = Trans i    (revM m)
revM (Instr i  m) = Instr i    (revM m)
revM (m1 :+: m2)  = revM m2 :+: revM m1
revM (m1 :=: m2)
  = let d1 = dur m1
        d2 = dur m2
    in if d1>d2
          then revM m1 :=: (Rest (d1-d2) :+: revM m2)
          else (Rest (d2-d1) :+: revM m1) :=: revM m2
```

Cutting a piece of music short

```
cut :: Dur -> Music -> Music

cut d m | d <= 0  = Rest 0
cut d (Note x d0) = Note x (min d0 d)
cut d (Rest d0)   = Rest (min d0 d)
cut d (m1 :=: m2) = cut d m1 :=: cut d m2
cut d (Tempo a m) = Tempo a (cut (d*a) m)
cut d (Trans a m) = Trans a (cut d m)
cut d (Instr a m) = Instr a (cut d m)
cut d (m1 :+: m2) =
        let m1' = cut d m1
            m2' = cut (d - dur m1') m2
        in m1' :+: m2'
```

Comments

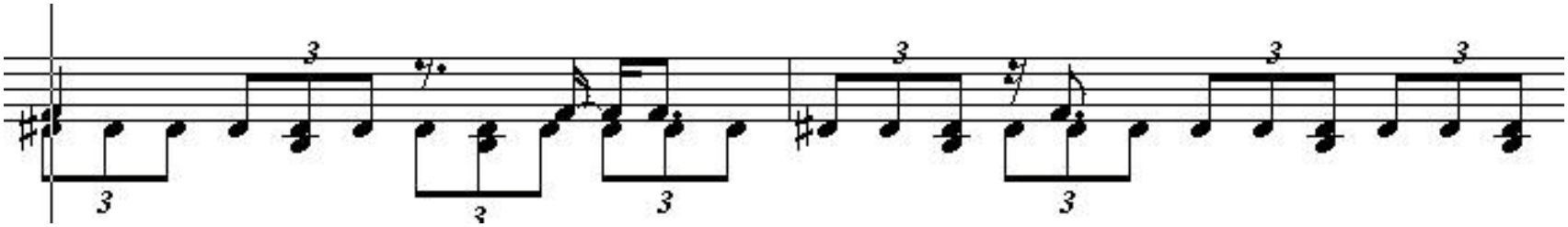
- `Music` is a high level abstract representation of music.
- Its analyzable so we can do many things with it
  - First, we can play it
  - But we can also
    - » compute its duration (without playing it)
    - » reverse it
    - » scale it's Tempo
    - » truncate it to a specific duration
    - » transpose it into another key

Percussion

```
data PercussionSound
  = AcousticBassDrum  -- MIDI Key 35
  | BassDrum1         -- MIDI Key 36
  | SideStick         -- ...
  | AcousticSnare | HandClap      | ElectricSnare | LowFloorTom
  | ClosedHiHat   | HighFloorTom  | PedalHiHat    | LowTom
  | OpenHiHat     | LowMidTom     | HiMidTom      | CrashCymbal1
  | HighTom       | RideCymbal1   | ChineseCymbal | RideBell
  | Tambourine    | SplashCymbal  | Cowbell       | CrashCymbal2
  | Vibraslap     | RideCymbal2   | HiBongo       | LowBongo
  | MuteHiConga   | OpenHiConga   | LowConga      | HighTimbale
  | LowTimbale    | HighAgogo     | LowAgogo      | Cabasa
  | Maracas       | ShortWhistle  | LongWhistle   | ShortGuiro
  | LongGuiro     | Claves        | HiWoodBlock   | LowWoodBlock
  | MuteCuica     | OpenCuica     | MuteTriangle
  | OpenTriangle      -- MIDI Key 82
    deriving (Show,Eq,Ord,Ix,Enum)
```

Let's beat the drums

```
perc :: PercussionSound -> Dur -> Music
perc ps = Note (pitch (fromEnum ps + 35))

funkGroove
  = let p1 = perc LowTom         qn
        p2 = perc AcousticSnare en
    in Tempo 3 (Instr Percussion (cut 8 (repeatM
         ( (p1 :+: qnr :+: p2 :+: qnr :+: p2 :+:
            p1 :+: p1 :+: qnr :+: p2 :+: enr)
           :=: roll en (perc ClosedHiHat 2) )
         )))
```



Music Presentation

- Music is a highlevel, abstract representation
- We call the playing of Music its Presentation
- Presentation requires “flattening” the Music representation into a list of low level events.
  - Events contain information about
    - » pitch
    - » start-time
    - » end-time
    - » loudness
    - » duration
    - » instrument etc.
- The MIDI standard is a file format to represent this low level information.
- Presentation is the subject of the next lecture.

MIDI Event List

Hours, Minutes, Seconds, Frames

Measure, Beats, Ticks

Pitch, Volume, Duration

Trk	HMSF	MBT	Ch	Kind	Data		
1	00:00:00:00	1:01:000	1	Note	C 4	127	2:000
1	00:00:01:00	1:03:000	1	Note	D 4	127	2:000
1	00:00:02:00	2:01:000	1	Note	E 4	127	2:000
1	00:00:03:00	2:03:000	1	Note	F 4	127	2:000
1	00:00:04:00	3:01:000	1	Note	G 4	127	2:000
1	00:00:05:00	3:03:000	1	Note	A 4	127	2:000
1	00:00:06:00	4:01:000	1	Note	B 4	127	2:000
1	00:00:07:00	4:03:000	1	Note	C 5	127	2:000

track

channel

Time in 2 formats