```
// Scan 32 note organ pedalboard and toe pistons and swell/crescendo with Leonardo board
// Using MIDIUSB library v.1.0.3
// John Harvey October 2017

// Loop time excluding delay 550us

//https://www.arduino.cc/en/Reference/MIDIUSB
//https://www.arduino.cc/en/Tutorial/MidiDevice
//https://github.com/arduino-libraries/MIDIUSB
//https://github.com/arduino/tutorials/blob/master/ArduinoZeroMidi/ArduinoZeroMidi.ino

#include <MIDIUSB.h>
//#include <pitchToFrequency.h>
//#include <pitchToNote.h>
//#include <frequencyToNote.h>

// Digital inputs to read pedal diode matrix columns
const byte Column0 = 0;
const byte Column1 = 1;
const byte Column2 = 2;
const byte Column3 = 3;
const byte Column4 = 4;
const byte Column5 = 5;
const byte Column6 = 6;
const byte Column7 = 7;

const byte Piston1 = 8;
const byte Piston2 = 9;
const byte Piston3 = 10;
const byte Piston4 = 11;

// Analogue inputs (10-bit ADC gives values 0-1023)
const byte SwellInput = A0;
const byte CrescendoInput = A1;

// Analog pins used as digital outputs to set pedal diode matrix rows
const byte Row0 = A2;
const byte Row1 = A3;
const byte Row2 = A4;
const byte Row3 = A5;

// Swell/Crescendo pedal range as found with actual devices
const byte SwellLow = 0x1B;
const byte SwellHigh = 0x4E;
const byte CrescendoLow = 0x00;  // tba
const byte CrescendoHigh = 0x7F; // tba

// Remember key states, true = key on, false = key off
boolean PedalKeyboardState[32]; // Max 32 C to G but St. Anne's Moseley only has 30 C to
F

// Remember toe piston states, true = depressed
boolean ToePistonState[4];

// Remember swell/crescendo pedal states
byte SwellState;
```

```
byte CrescendoState;

void setup() {

  pinMode (Column0, INPUT_PULLUP);
  pinMode (Column1, INPUT_PULLUP);
  pinMode (Column2, INPUT_PULLUP);
  pinMode (Column3, INPUT_PULLUP);
  pinMode (Column4, INPUT_PULLUP);
  pinMode (Column5, INPUT_PULLUP);
  pinMode (Column6, INPUT_PULLUP);
  pinMode (Column7, INPUT_PULLUP);

  pinMode (Piston1, INPUT_PULLUP);
  pinMode (Piston2, INPUT_PULLUP);
  pinMode (Piston3, INPUT_PULLUP);
  pinMode (Piston4, INPUT_PULLUP);

  pinMode (Row0, OUTPUT);
  pinMode (Row1, OUTPUT);
  pinMode (Row2, OUTPUT);
  pinMode (Row3, OUTPUT);
  digitalWrite(Row0, HIGH);
  digitalWrite(Row1, HIGH);
  digitalWrite(Row2, HIGH);
  digitalWrite(Row3, HIGH);

  pinMode(LED_BUILTIN, OUTPUT); // For testing only

  for (byte note = 0; note <= 31; note ++) {
    PedalKeyboardState[note] = false;  // All notes off
  }

   for (byte toePiston = 0; toePiston <= 3; toePiston ++) {
    ToePistonState[toePiston] = false;
  }

  SwellState = 127;   // Swell box fully open
  CrescendoState = 0; // Crescendo at minimum, no stops selected

}

void loop() {

// SCAN PEDALS (MIDI channel 3)

  for (byte row = 0; row <= 3; row ++) {

    if (row == 0) {
      digitalWrite(Row0, LOW);
      digitalWrite(Row1, HIGH);
      digitalWrite(Row2, HIGH);
      digitalWrite(Row3, HIGH);
    }

    if (row == 1) {
```

```
      digitalWrite(Row0, HIGH);
      digitalWrite(Row1, LOW);
      digitalWrite(Row2, HIGH);
      digitalWrite(Row3, HIGH);
    }

    if (row == 2) {
      digitalWrite(Row0, HIGH);
      digitalWrite(Row1, HIGH);
      digitalWrite(Row2, LOW);
      digitalWrite(Row3, HIGH);
    }

    if (row == 3) {
      digitalWrite(Row0, HIGH);
      digitalWrite(Row1, HIGH);
      digitalWrite(Row2, HIGH);
      digitalWrite(Row3, LOW);
    }

    byte keyNumber;
    boolean PreviousKeyState;
    boolean CurrentKeyState;

    for (byte col = 0; col <= 7; col ++) {

      keyNumber = (row * 8) + col;
      PreviousKeyState = PedalKeyboardState[keyNumber];
      CurrentKeyState = !digitalRead(col);  // Low = key on, so invert;

      if ((PreviousKeyState == false) && (CurrentKeyState == true)) {  // Note on
        PedalKeyboardState[keyNumber] = true;
        midiEventPacket_t noteOn = {0x09, 0x92, byte(keyNumber +0x24), 0x40}; //
keyNumber +0x24 produces an int in compiler so use byte() to reconvert to byte value
otherwise compiler outputs a narrowing conversion warning
        MidiUSB.sendMIDI(noteOn);
        MidiUSB.flush();
      }

      if ((PreviousKeyState == true) && (CurrentKeyState == false)) {  // Note off
        PedalKeyboardState[keyNumber] = false;
        midiEventPacket_t noteOff = {0x08, 0x82, byte(keyNumber + 0x24), 0x40};
        MidiUSB.sendMIDI(noteOff);
        MidiUSB.flush();
      }
    }
  }

  digitalWrite(Row0, HIGH);
  digitalWrite(Row1, HIGH);
  digitalWrite(Row2, HIGH);
  digitalWrite(Row3, HIGH);

// TOE PISTONS (MIDI channel 3) using D8-11 as digital inputs to keep wiring separate
from pedals
// Only 4 inputs so no need for row/column matrix diodes, just wire contacts to ground
```

```
and D8-11
// In Hauptwerk use Input: Momentary piston: MIDI note-on (i.e.toggle) and set Hauptwerk
toe pistons to first 4 notes above pedals:
// Swell to Great:  G3# (068, #44)  Hauptwerk 035 Coupler: Sw to Gt
// Swell to Pedal:  A3  (069, #45)  Hauptwerk 033 Coupler: Sw to Ped
// Great to Pedal:  A3# (070, #46)  Hauptwerk 032 Coupler: Gt to Ped
// Trombone 16:     B3  (071, #47)  Hauptwerk 007 Stop: Ped: Trombone 16

for (byte piston = 0; piston <= 3; piston ++) {

    boolean PreviousPistonState;
    boolean CurrentPistonState;

    PreviousPistonState = ToePistonState[piston];
    CurrentPistonState = !(digitalRead(piston + 8));  // Low = piston depressed, so
invert;

    if ((PreviousPistonState == false) && (CurrentPistonState == true)) {
      ToePistonState[piston] = true;
      midiEventPacket_t noteOn = {0x09, 0x92, byte(piston + 0x24 + 32), 0x40};
      MidiUSB.sendMIDI(noteOn);
      MidiUSB.flush();
    }

    if ((PreviousPistonState == true) && (CurrentPistonState == false)) {
      ToePistonState[piston] = false;
    }
  }

// SWELL AND CRESCENDO PEDALS
// Swell pedal fully open LDR = 16K, fully closed LDR = 2.8K

//  byte SwellValue = analogRead(SwellInput) / 8; // Integer division gives range 0-127
//  if (SwellValue != SwellState) {
//    SwellState = SwellValue;
//    midiEventPacket_t event = {0x0B, 0xB0, 0x07, SwellValue};
//    MidiUSB.sendMIDI(event);
//    MidiUSB.flush();
//  }

  byte SwellValue = analogRead(SwellInput) / 8; // Integer division gives range 0-127
  if (SwellValue != SwellState) {
    SwellState = SwellValue;
    byte ModifiedSwellValue = (SwellValue * 127 / (SwellHigh - SwellLow)) - SwellLow;
// Expand limited swell pedal travel to full 0-127 range
    midiEventPacket_t event = {0x0B, 0xB0, 0x07, ModifiedSwellValue};
    MidiUSB.sendMIDI(event);
    MidiUSB.flush();
  }

//  byte CrescendoValue = analogRead(CrescendoInput) / 8; // Integer division gives
range 0-127
//  if (CrescendoValue != CrescendoState) {
//    CrescendoState = CrescendoValue;
//    midiEventPacket_t event = {0x0B, 0xB0, 0x09, CrescendoValue};
//    MidiUSB.sendMIDI(event);
```

```
//    MidiUSB.flush();
//  }

  byte CrescendoValue = analogRead(CrescendoInput) / 8; // Integer division gives range
0-127
  if (CrescendoValue != CrescendoState) {
    CrescendoState = CrescendoValue;
    byte ModifiedCrescendoValue = (CrescendoValue * 127 / (CrescendoHigh -
CrescendoLow)) - CrescendoLow;  // Expand limited crescendo pedal travel to full 0-127
range
    midiEventPacket_t event = {0x0B, 0xB0, 0x09, ModifiedCrescendoValue};
    MidiUSB.sendMIDI(event);
    MidiUSB.flush();
  }

  delay(1);

}
```