```
// Scan two 61 note organ manuals and 32 note pedals with Leonardo board
// Using MIDIUSB library v.1.0.3
// John Harvey September 2017

//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>

// Arduino input pins to read keyboard 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;

// Arduino output pins to drive 74138 A0-2
const byte RowSelect0 = 8;
const byte RowSelect1 = 9;
const byte RowSelect2 = 10;

// Arduino output pins, low to select manual/pedals
const byte SelectSwell = 11;
const byte SelectGreat = 12;
const byte SelectPedal = 13;

// Remember key states, true = key on, false = key off
boolean SwellKeyboardState[64]; // Only 61 keys but row/column scan of 8x8 keyboard matrix goes from 0 to 63
boolean GreatKeyboardState[64];
boolean PedalKeyboardState[32]; // Max 32 C to G but St. Anne's Moseley only has 30 C to F

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 (RowSelect0, OUTPUT);
  pinMode (RowSelect1, OUTPUT);
  pinMode (RowSelect2, OUTPUT);
  digitalWrite(RowSelect0, LOW);
  digitalWrite(RowSelect1, LOW);
  digitalWrite(RowSelect2, LOW);

  pinMode (SelectSwell, OUTPUT);
  pinMode (SelectGreat, OUTPUT);
  pinMode (SelectPedal, OUTPUT);
  digitalWrite(SelectSwell, HIGH);
  digitalWrite(SelectGreat, HIGH);
  digitalWrite(SelectPedal, HIGH);

  for (byte note = 0; note <= 63; note ++) {
    SwellKeyboardState[note] = false;  // All notes off
    GreatKeyboardState[note] = false;  // All notes off
  }

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

}

void loop() {

// Kept code inline rather than in separate functions to keep scan time to a minimum; scan
of swell + great + pedals down to 1.4ms from over 2ms before careful optimisation

// SCAN SWELL (MIDI channel 1)
  digitalWrite(SelectSwell, LOW);

  for (byte row = 0; row <= 7; row ++) {
    digitalWrite(RowSelect0, row & B00000001);
    digitalWrite(RowSelect1, row & B00000010);
    digitalWrite(RowSelect2, row & B00000100);

    byte keyNumber;
    boolean PreviousKeyState;
    boolean CurrentKeyState;

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

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

      if ((PreviousKeyState == false) && (CurrentKeyState == true)) {  // Note on
        SwellKeyboardState[keyNumber] = true;
        midiEventPacket_t noteOn = {0x09, 0x90, keyNumber + 0x24, 0x40};
        MidiUSB.sendMIDI(noteOn);
        MidiUSB.flush();
      }

      if ((PreviousKeyState == true) && (CurrentKeyState == false)) {  // Note off
        SwellKeyboardState[keyNumber] = false;
        midiEventPacket_t noteOff = {0x08, 0x80, keyNumber + 0x24, 0x40};
        MidiUSB.sendMIDI(noteOff);
        MidiUSB.flush();
      }
    }
  }

  digitalWrite(SelectSwell, HIGH);

// SCAN GREAT (MIDI channel 2)
  digitalWrite(SelectGreat, LOW);

  for (byte row = 0; row <= 7; row ++) {
    digitalWrite(RowSelect0, row & B00000001);
    digitalWrite(RowSelect1, row & B00000010);
    digitalWrite(RowSelect2, row & B00000100);

    byte keyNumber;
    boolean PreviousKeyState;
    boolean CurrentKeyState;

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

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

      if ((PreviousKeyState == false) && (CurrentKeyState == true)) {  // Note on
        GreatKeyboardState[keyNumber] = true;
        midiEventPacket_t noteOn = {0x09, 0x91, keyNumber + 0x24, 0x40};
        MidiUSB.sendMIDI(noteOn);
        MidiUSB.flush();
      }

      if ((PreviousKeyState == true) && (CurrentKeyState == false)) {  // Note off
        GreatKeyboardState[keyNumber] = false;
        midiEventPacket_t noteOff = {0x08, 0x81, keyNumber + 0x24, 0x40};
        MidiUSB.sendMIDI(noteOff);
        MidiUSB.flush();
      }
    }
  }

  digitalWrite(SelectGreat, HIGH);

// SCAN PEDAL (MIDI channel 3)
  digitalWrite(SelectPedal, LOW);
  for (byte row = 0; row <= 3; row ++) {
    digitalWrite(RowSelect0, row & B00000001);
    digitalWrite(RowSelect1, row & B00000010);
    digitalWrite(RowSelect2, row & B00000100);  // Not used but need to set 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, keyNumber + 0x24, 0x40};
        MidiUSB.sendMIDI(noteOn);
        MidiUSB.flush();
      }

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

  digitalWrite(SelectPedal, HIGH);

}
```