# Assignment #5

CS 245, Spring 2018

*Due Thursday, February 15*

## Before you start the assignment

You will need to install and test some programs before you start on the programming assignment. Here is a checklist for these tasks.

- PortMidi — cross–platform MIDI API

  ☐ Install PortMidi on your machine. If you are using Visual Studio on a Windows machine, you can use the pre–compiled version that is available on the CS 245 class page. Otherwise, you will need to download it from

  `http://portmedia.sourceforge.net/portmidi/`

  ☐ Test PortMidi. Compile and run the program `midi_out_pm.cpp`. Using the Visual Studio 2015 command prompt,

  ```
  cl /EHsc midi_out_pm.cpp portmidi.lib
  ```

  will compile the program. To obtain MIDI device information, invoke the program with no command line arguments:

  ```
  midi_out_pm
  ```

  You will get a list of available MIDI devices (both input and output devices). E.g.,

  ```
  0: Microsoft MIDI Mapper
  1: Microsoft GS Wavetable Synth
  ```

  (the output that you get might be different). Now run the program with command line parameters to send some MIDI messages to a selected MIDI output device. For example,

  ```
  midi_out_pm 1 57 0
  ```

  The first parameter (1) is the output device number (choose from above list), the second parameter (57) is a MIDI note index, and the third parameter (0) is the patch number. Invoking the program with command line parameters should (hopefully) result in a note being played. The note should continue until you hit the ENTER key. Try out different parameter values.

- Pthreads–Win32 — POSIX thread API for Win32

  ☐ Make sure that you can use POSIX threads on your machine with your compiler. If you are using Windows and Visual Studio, you can use the pre–compiled version that is available on the CS 245 class page. If you are running Linux, you do not need to do anything.

  ☐ Compile and run the test program `pthread_test.cpp`. From the MSVC command prompt, you can compile the test with

  ```
  cl /EHsc pthread_test.cpp portmidi.lib pthreadVC2.lib
  ```

  Again, running the program without any argument will produce a list of available MIDI devices. Run the program with using a single argument: the selected device number. You should hear a random sequence of annoying tones. Hit the ENTER key to make it stop.

- Virtual Piano MIDI Keyboard (VMPK) — computer keyboard as a MIDI device

  ☐ Download and install the Virtual Piano MIDI Keyboard (VMPK) program from this site:

  `http://vmpk.sourceforge.net/`

  ☐ Run VMPK. Connect the MIDI output of VMPK to the MS wavetable software synthesizer. To do this, look under Edit→MIDI Connections. Select "Windows MM" from the "MIDI OUT Driver". Now select "Microsoft GS Wavetable Synth" from "Output MIDI Connection". You should now be able to hear notes when you play them.

- loopMIDI — MIDI virtual port server

  ☐ Download and install the loopMIDI program either from the CS 245 class page, or directly from this site:

  `http://www.tobias-erichsen.de/software/loopmidi.html`

  ☐ Run loopMIDI. Create a virtual MIDI port by clicking on the + button (you can create and remove multiple virtual ports using this program, but we will only need one).

  ☐ Connect the MIDI output of VMPK to the loopMIDI port that you just created by selecting "loopMIDI port" from "Output MIDI Connection", as described above. In the loopMIDI window, you should see data being received whenever you play notes using VMPK.

  ☐ Compile the `midi_intercept_pm.cpp` program from class (the source file is available on Moodle). From the MSVC 2015 command prompt, this is:

  ```
  cl /EHsc midi_intercept_pm.cpp portmidi.lib pthreadVC2.lib
  ```

- ☐ Run the `midi_intercept` program from a command prompt to connect VMPK (as input device via the loopMIDI port) to the GS wavetable synthesizer (as output device). When you play notes on VMPK, MIDI messages should now be displayed on the command line.

- MIDI message format.

  ☐ Familiarize yourself with the page:

  `https://www.midi.org/specifications/category/reference-tables`

  In particular, look at the "Summary of MIDI Messages" and "Control Change Messages" links.

# Your task

I will provide you with the header file `MidiIn.h`, whose contents are as follows.

```
#include <string>
#include "pthread.h"
#include "portmidi.h"

class MidiIn {
  public:
    static std::string getDeviceInfo(void);
    MidiIn(int devno);
    void start();
    void stop();
    virtual ~MidiIn(void);
    virtual void onNoteOn(int channel, int note, int velocity) { }
    virtual void onNoteOff(int channel, int note) { }
    virtual void onPitchWheelChange(int channel, float value) { }
    virtual void onVolumeChange(int channel, int level) { }
    virtual void onModulationWheelChange(int channel, int value) { }
    virtual void onControlChange(int channel, int number, int value) { }
  private:
    PmStream *input_stream;
    bool process_events,
         thread_running;
    pthread_t event_thread;
    static void* eventLoop(void *vp);
};
```

This defines a framework class for handling MIDI input from a particular input device. You are to implement the *non–virtual* functions in this class, as well as the (base class virtual) destructor. You should not implement the virtual functions. A user uses this class by sub–classing it and overriding the virtual functions with their own specific actions. The functions that you should implement are described below.

`getDeviceInfo()` — this function should return an indexed list of all available MIDI input devices. The first device should be labeled 0, the second 1, and so on. E.g.,

```
0: loopMIDI Port
1: loopMIDI Port 1
```

`MidiIn(devno)` — (constructor) opens the MIDI input device with index number `devno`. If the the device cannot be opened a `runtime_exception` should be thrown. This is a standard exception declared in `stdexcept`. In your implementation, you will need to (1) initialize PortMidi, (2) open a PortMidi stream for MIDI input, and (3) create a thread to poll for MIDI messages.

`start()` — starts/restarts processing of MIDI input events. Polling for messages in the thread function should only occur after this function is called.

`stop()` — stops/pauses processing of MIDI input events. Polling for messages in the thread function should stop until the next call to `start`.

`~MidiIn()` — (destructor) closes the input device. In your implementation, you will need to (1) stop the thread and wait for it to finish, (2) close the PortMidi stream, and (3) terminate PortMidi.

`eventLoop(vp)` — thread function function used to poll for MIDI input messages, and process any events as they arrive. This functions should continuously poll for input messages until the class destructor is called. If a MIDI input message is present, the message should be deciphered, and the appropriate (virtual) hook function called. Note that this function is (and must be) *static,* so you do not have direct access to the non–static members of the `MidiIn` class! To get around this inconvenience, you will have to pass the `this` pointer as data when you create the thread. The void pointer `vp` can then be recast to the class pointer.

In implementing the `eventLoop` thread function, you will decipher the MIDI input message and call the appropriate virtual function.

`onNoteOn` — called when a note is turned on.

`onNoteOff` — called when a note is turned off. This may occur if the note–off message is received, or if a note–on message with 0 velocity is received.

`onPitchWheelChange` — called when the pitch wheel position is changed. The MIDI message received uses two bytes to specify the pitch shift value. However, only the first seven bits in each byte are used, so that the value is an unsigned 14–bit value in the range 0 to $2^{14} - 1$ with midpoint $2^{13}$. You will convert this to a floating point value in the range $-1 \leq$ `value` $\leq 1$. For example, if the first data byte has a value of $b_1 = 14$ $(Eh)$, and the second data byte has the value $b_2 = 20$ $(14h)$, then the corresponding 14 bit integer is

$$20 \cdot 2^7 + 14 = 2574$$

Using a midpoint of $2^{13} = 8192$, the floating point that would be passed is then

$$(2574 - 8192) \cdot \frac{1 - (-1)}{2^{14} - 1} \approx -0.6858$$

`onVolumeChange` — called when the input device's volume control has been changed. This is actually a MIDI *control change* message (Bh) with control number 7h. So in addition to calling the `onVolumeChange` function, the `onControlChange` function should be called as well (see below).

`onModulationWheelChange` — called when the input device's modulation wheel position is changed. Since this is also a MIDI control change message, the `ControlChange` function should also be called.

`onControlChange` — called when a control message has been received from the input device.

Your submission for this assignment should consist of a single source file, named `MidiIn.cpp`. You may include any standard C++ header file, as well as the header file `MidiIn.h` (which includes `pthread.h` and `portmidi.h`).