EE 553 - Engineering C++

Final Design Document:

Music Editor Project

May 15, 2016

Marty Marshale

Caroleen Chen

Kyria Johnson

Erwei Zhu

Nicole Regenauer

Jack Liang

Bryan Charalambous

**Team Project Description**

The goal of this project is to create a Music Editor using C++ and Qt. This program is able to successfully open MIDI files, convert the file to notation, display the MIDI file, and export it to a PDF. A basic user interface was created that gives the user the option to open a file, save a file, or exit. This program runs on Windows or Mac machines.

The project was split into four sub-teams in charge of different tasks:

1. User Interface that was in charge of creating the basic user interface and connecting all the classes so that the program ran.
2. Reading the MIDI files to parse data from the input into an array of notation information to print out a file.
3. Converting the file to notation and displaying the MIDI files as sheet music so that it was possible for the user to read the notes.
4. Translating MIDI files to PDF so that it was possible to save the file.

**Major Classes Description**

**Craig Sapp’s midifile:** Open source library that provides easy-to-use interfaces for parsing data from MIDI files. Used to get specific pieces of data from MIDI messages and header data.

**drawStaff:** This class facilitates in drawing the music notation in the GUI using a vector of “notes” output by the accompanying readMidi class

* Methods:
* newMeasure(bool isBegin=false,bool isEnd =false): creates a new measure after

the total number of notes/beats for a measure in the timesignature has been reached

* renderNote(int xpos, int typos, int ind, int yoff, string dur, int sharpOrFlat=0):

render notes, rather than copying the same code for every single note case.

* drawNotes(string note , int octave , string duration, int startIndex):

use the renderNote function to draw notes on the newly created messure

* vector<note> buildNoteArray(): iterate through to render the music note by note

**PDFExport:** This class reads the vector of “notes” provided by the drawStaff class and uses the data to replicate the music notation and export it into a pdf file

* Methods:
  + drawStaffLine(int xs, int ys): renders a single block of staff lines, with the clef included
  + addTimeSig(int xs, int ys): adds a time signature to the beginning of a staff line
  + drawPage(int pos\_x, int pos\_y): draws all staff lines and key signatures on new page, starting at specified x and y position
  + addNotes(int xs, int ys, string note, int octave, QString duration, int startIndex): adds a new note to staff in appropriate location using specified font
  + completeDrawing() : frees some memory allocated by Qt classes
  + readVector(vector<note> notes): reads through note vector created by notation team and renders notes to staff in the appropriate locations, navigating to a new line/measure and creating a new page when necessary

**readMidi:** This class iterates through note events in the midi file to output an array of notes in sequential order to be used by drawStaff to render notation to the GUI and used by pdfedport to render to PDF

* Methods:
* int getKeySig(): get the key signature from the MIDI file
* string getTimeSig(): get the time signature from the MIDI file
* getNotePos(int startTime): logic to figure out the horizontal index position of a

note or reset

* string rhythmType(double duration): determines the rhythmtype of a note ,eigth

note

* vector<note> builArray(): iterates through the events in the midifile to return a

vector of notes

**MidifileREAD:** This class is for reading the data of a midi file in byte, and separating the data into a vector for the future use.

* Methods:
* Getdata():read the data of a midi fie in byte for the next four class to separate

the data

* Getsize(): get the size of a midi file for the use of the future loop

**Header**: This class is for reading the Header Chunk data of a midi file. A Header Chunk is structured as: MThd + Header length + Format + Track Number + Tick Speed. Seperate the data to “Format”, “TrackNumber” and “TickSpeed” with the mark "\r\n".

* Methods:
* GetFormat(): Read the Format and print out in hex for checking and using
* GetTrackNumChar():Read the TrackNumber and print out in hex for checking and

using

* GetDIV():Read the TickSpeedand print out in hex for checking and using
* GetTrackNum():Read the TrackNumber in int for the future loop to separate the

track message

* GetTick():Read the TickSpeed in int to calculate the Time and print out in the

notation file for check

**Track**: This class is for reading the Track Chunk data of a midi file. A Track Chunk is structured as: MTrk + Track length + Track Event. Separate each tracks with the mark "\r\n"

* Methods:
* GetTrackLengthChar(): Read the Track length in byte for checking
* GetTrackEvent(): get track event in byte for checking
* GetTrack(): get track in byte, so I can use this data to print out the tracks with

the lists of their own track messages

* GetTrackLength(): Read the Track length in int, so I can use this to determine the

loop for separating the track messages

**Notation\_Event**: This class is for reading the Notation of a midi file. A Notation is structured as: Speed + command nibble + Data. Speed is determined as VLV:80~8F + length or length. Separate each status with the mark "\r\n". Separate tracks with the mark "00 2F FF 00"

* Methods:
* GetVLVTime(int index, int &vlv): Calculate the speed in int with the VLV Rules
* WriteByteTwo(int index, int tick): Read the message with speed and note name
* WirteByteThree(int index, int tick): Read the message with speed, note name

and one byte data

* WriteByteFour(int index, int tick):Read the message with speed, note name and

two bytes data

* WriteByteChord(int index, int tick): Read the chord only with pitch and velocity
* WriteByteF0(int index, int tick, int length): Write the message of Meta Type F0
* WriteByteFF(int index, int tick, int length): Write the message of Meta Type FF
* GetNotation(): Save the notation list to print out

**Seperate\_Event**: This class is for reading the Meta data of a midi file. Meta is structured as: Speed + FF + Meta type + Meta length + Meta Event. Separate the data with the mark “|”.

* Methods:
* GetMetaListChar():get Meta meesage in byte, so I can use this data to print out

with the Meta Types with their own message

**Documentation**

**Prerequisites:**

1. You must have the correct font(“Norfolk.otd”) installed to successfully see the notes in the program.
2. Make sure to have all the correct files:
   1. main.cpp, mainWindow.cpp, mainWindow.h, musicEditor.pro
   2. drawStaff.h, pdfexport.h, readHex.h, readMidi.h
   3. chartostring.hh, midi.hh
   4. Craig Sapp’s midifile library components

Source code is available at our git repository.

<https://github.com/StevensDeptECE/EE553_Music_Editor>

**Compiling Instructions:**

**Easy Way:**

1. Install Qt for your system
2. Run Qt Creator and navigate to this folder in the source tree
3. From the file menu, open the "musicEditor.pro" file and use the default options to open as a project
4. Build and run from Qt Creator

**Hard Way (Windows):**

1. On Windows: Download, install, and update MSYS2 - <http://www.msys2.org/>
2. Install the mingw toolchain for your host system as well as qt5:
   1. For mingw, example install looks like this: "pacman -S mingw-w64-x86\_64-toolchain" (Just hit enter to install all components)
   2. For qt5, the command looks like this: "pacman -S mingw-w64-x86\_64-qt5"
3. Open the "MSYS MinGW x64" (or if you’re stuck in 2005, x86) and cd to this directory
4. Run qmake
5. Run mingw32-make
6. Output binary should be in the debug folder

**Hard Way (MAC OSX):**

1. Ensure that XCode is installed on your system
2. Install the latest XCode command line tools package from the Apple Developer’s portal

To check whether the command line tools package is correctly installed open the terminal application on your system and run the following command “xcode-select -p”.

The command will return a ‘2’ if the gcc and make configurations are not found, a ‘0’ indicates that the tools are present

1. Run qmake
2. Output binary should be in the debug folder