

Conversion of a Music Macro Language **Variant to MIDI**

Analysis

Background to Problem

The MIDI file format is a widespread and established standard for storing music. The music macro language (MML) is a niche language for describing music in a comparable way to how music is stored in MIDI files. It has no real standards, but has evolved over the years into different versions, the use of which are provided by BASIC implementations. One of the key issues with the MML is how there is not much software available that can play it; another issue is that the MML is, by design, only suited for monophonic tracks.

This project seeks to tackle both of these issues by providing a facility that can convert a number of files containing a music description language similar to MML into a single multi-channel MIDI file. This allows the benefits of both the MML and MIDI formats to be taken advantage of – music is easily entered with MML, and it is easily playable using the MIDI format.

Research Methods

The MIDI Specification

The first thing I have researched for this project is the specification of MIDI files.

Values in MIDI files are either stored as big endian binary numbers or as big endian variable length quantities.

MIDI files are composed of “chunks”, which all consist of, at least, a 4-byte string, identifying the type of chunk, and the length of the chunk as a 32-bit integer. There are two different types of chunks featured in the MIDI specification 1.1 – header chunks and track chunks.

The following table shows how headers chunks are structured:

| Bits 0-7 | Bits 8-15 | Bits 16-23 | Bits 24-31 |
|-----------------------------|-----------|------------|------------|
| The ASCII characters “MThd” | | | |
| Length | | | |
| Format | | Ntrks | |
| Division | | | |

“Length” is a 32-bit unsigned integer describing how many bytes are left of the chunk. It is always 6 for a header chunk.

“Format” is a 16-bit unsigned integer that describes the format of the MIDI file. It can currently take three values:

| Format | Description |
|--------|--|
| 0 | The MIDI file is composed of a header chunk and a single track chunk |
| 1 | The MIDI file is composed of a header chunk and multiple track chunks that are to be played simultaneously |
| 2 | The MIDI file is composed of a header chunk and multiple track chunks that are to be played independently |

“Ntrks” is a 16-bit unsigned integer that indicates how many track chunks there are in the file.

“Division” is a 16-bit unsigned integer that sets the meaning of the delta-times (which are variable length quantities put before every event, representing how long should be left from the end of the last event to the beginning of the current one).

The next table shows how track chunks are structured:

| Bits 0-7 | Bits 8-15 | Bits 16-23 | Bits 24-31 |
|-----------------------------|-----------|------------|------------|
| The ASCII characters “MTrk” | | | |
| Length | | | |
| MTrk Events ... | | | |

“Length”, like in the header chunk, is a 32-bit unsigned integer describing how many remaining bytes there are to be read in the chunk.

The rest of the track chunk consists of MTrk events, which are structured as follows:

<MTrk event> = <delta time> <event>

“Delta time” is a variable length quantity that represents the length of time to leave between this event and the last one.

“Event” can be one of three different varieties: MIDI, sysex and meta.

MIDI events are MIDI channel messages, such as note down and note up.

Sysex (system exclusive) events are messages that are sent via the MIDI file, but are not related to the playing of it.

Meta events describe how the MIDI commands should be played, and give supplementary information about the track, such as its name and cues.

The MIDI events that will be relevant for the project are shown in the table below: “nnnn” is where the nibble representing the channel number the message is for is placed.

| Status Byte | Data Bytes | Description |
|-------------|------------|--|
| 1000 nnnn | 0kkkkkkk | Note off event – sent when a key is released. 0kkkkkkk is the note number. 0vvvvvvv is the velocity. |
| | 0vvvvvvv | |
| 1001 nnnn | 0kkkkkkk | Note on event – sent when a key is depressed. 0kkkkkkk is the note number. 0vvvvvvv is the velocity. |
| | 0vvvvvvv | |
| 1100 nnnn | 0ppppppp | Patch change – sent when the instrument is changed. 0ppppppp is the new patch number. |

The meta events that will be used in the project are shown below.

| Form (hex) | Description |
|-----------------------|--|
| FF 03 [length] [name] | Track name – this command contains the name of the track. [length] is the length of the name following, as a variable length quantity. [name] is an ASCII string. |
| FF 2F 00 | End of track – always present at the end of track chunk. |
| FF 51 03 tttttt | Set tempo. “tttttt” is three bytes that represent the number of microseconds per MIDI crochet. |
| FF 58 04 nn dd cc bb | Time signature. “nn” is a byte representing the top of a traditional time signature. “dd” is a byte representing the bottom of a traditional time signature as 2^{dd} . “cc” is a byte setting the number of MIDI clocks per metronome tick. “bb” is a byte setting the number of demisemiquavers per 24 MIDI clocks (usually 8) |

A full example of a MIDI file in hexadecimal is shown below, split up into a table with descriptions on each section.

| | |
|-------------------------|--|
| 4D 54 68 64 | "MThd" |
| 00 00 00 06 | Length of header chunk |
| 00 00 | Format 0 |
| 00 01 | One track |
| 00 20 | 32 ticks per crochet |
| 4D 54 72 6B | "MTrk" |
| 00 00 00 2A | Length of track chunk |
| 00 FF 58 04 04 02 18 08 | Set time signature to 4/4. Set 24 MIDI clocks per metronome tick. Set 8 demisemiquavers per 24 MIDI clocks |
| 00 FF 51 03 07 A1 20 | Set tempo to 0x07A120 (500 000) microseconds per 24 MIDI clocks. This corresponds to 120 BPM in this context. |
| 00 C0 30 | Set channel 0 to patch number 48 |
| 00 C1 30 | Set channel 1 to patch number 48 |
| 00 90 45 7F | Play note with MIDI number 69 (A4) on channel 0 with maximum velocity. |
| 81 10 80 45 7F | Release note with MIDI number 69 (A4) on channel 1 with maximum velocity after a delta time of 0x90. |
| 00 91 40 7F | Play note with MIDI number 64 (E4) on channel 1 with maximum velocity. |
| 7F 80 40 7F | Release note with MIDI number 64 (E4) on channel 1 with maximum velocity after a delta time of 0x7F. |
| 00 FF 2F 00 | End of track |

Development Tools and Method

There are many programming languages that would be suitable for this project, but I have chosen to go with C, as I am comfortable with it, and I feel that it is suitable, being a low-level language and the amount of byte-wise manipulation needed in this project.

Python would also have been a good choice because of how easy string manipulation is in it, which would make the processing of the input language more convenient.

I found two tools in my research that I have decided to use in this project to produce the parsing portions of the code – Lex and Yacc. Lex takes a C file containing regular expressions and code to execute upon matches, and Yacc, which works in conjunction with Lex, takes a C file containing Back-Naur form grammar and code to execute upon reductions.

To learn how to use Lex and Yacc, I've purchased an O'Reilly book on them, which I'll be using throughout the project. I anticipate that I'll be using them in a fairly conventional manner, so the examples provided in the book will likely prove very useful.

An alternative to using Lex and Yacc would be writing my own parser and lexer, which could potentially yield faster code, but would take a great deal longer to develop.

To compile my code, I will use makefiles in combination with GCC. This is a common practice for C development, and one I am comfortable with. The alternative is entering the GCC, Lex and Yacc calls every time compilation is performed, which could be very inconvenient.

Clang is another C compiler that could be used, which has many advantages over GCC, such as compilation speed, but I am used to using GCC and have it already available on the machines I will be developing the project on.

Description of the Current System

The current way to convert multiple music macro language like files into a single multi-channel MIDI file is to step through by hand and enter the information as if writing a new MIDI file. This is tedious, unreliable and slow.

Identification of Potential Users and User Needs

There is a small, but dedicated, remaining group of musicians that still use the music macro language. I reached out to some of these musicians online and asked what features they would value most from project.

The most requested feature was that the solution should have an easy to use interface, which doesn't require any technical knowledge to use.

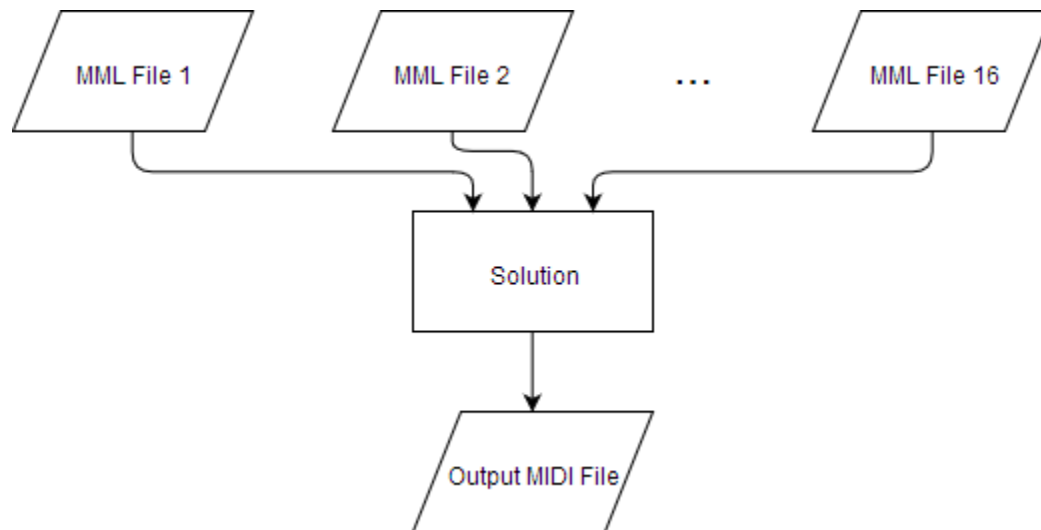
Another requested feature was that the MML version taken as an input to the solution should have at least the functionality of existing versions, and have similar syntax. If even one feature were removed it may impede the user's workflow.

Finally, another common request was the solution should be fast enough to keep up with a musicians work flow, otherwise this would be stifling for their creativity.

I have used the information I gained from corresponding with the potential users in order to create the objectives for the project, which are covered in a following section.

Data Flow Diagrams

The following flow chart shows how the solution to the problem should work in terms of file flow:



Use Cases

A musician whom uses the MML to compose music would likely find the tools written for this project very useful, as it allows the conversion of their preferred format to an easily playable and portable form.

The solution would also very suitable for a less musically experienced individual, as it is easy to alter and enter music in MML, in contrast to traditional sheet music.

Potential Solutions

One solution to the problem is to have one terminal program that the user enters the paths to the input MML files into, and have it output the desired MIDI file. There are a number of pros and cons in this potential solution:

| Single Program Advantages | Single Program Disadvantages |
|---------------------------|---|
| Easy to use | The program would be complex, given what functionality it must have |

Another solution would be to have two programs that are designed to be used in conjunction: the first would take a single MML file and output a single-channel MIDI file, and the second would take multiple single-channel MIDI files and output a single multiple-channel MIDI file. Some advantages and disadvantages to this approach are shown in the following table.

| Program Pair Advantages | Program Pair Disadvantages |
|--|----------------------------|
| Each program would be more simple to write, compared to a single complex program | More difficult for to use |
| Each channel of the final MIDI file can be listened to, as a single-channel MIDI file of it would be generated before-hand | |

Chosen Solution

I have chosen to use the solution featuring a pair of programs. This is primarily because I feel that the complexity of both programs combined would be less than the complexity of one larger program.

Objectives of the Project

- A program should be written that takes text file containing a variant of the MML as an input, and outputs a single-channel MIDI file that can be play with conventional software.
- A program should be written that takes up to 16 MIDI files generated by the aforementioned program and combines them into a single multi-channel MIDI file. This combined MIDI file should be playable using conventional software also.
- The programs written for this project should:
 - Complete their execution in under one second, so as to not interrupt the users work flow
 - Use less than half a megabyte of memory during execution
 - Be a less than a quarter of a megabyte in size
- A version of the music macro language should be designed that will be used as the input for the program that generates a single-channel MIDI file from a single MML file. This language should:
 - Have all the functionality of existing variants of the MML, including support for:
 - Octave changing
 - Accidentals
 - Default length setting
 - Volume changing
 - Tempo setting
 - Macros

Documented Design

Target Hardware

The programs written for this project will be targeted at Unix based operating systems such as OS X and Linux, as this is what I will be developing them on, and I am used to working with these systems. Also, the programs will be written to run on little endian processors – this is relevant because the endianness of many values will need to be flipped during the generation of the MIDI files.

Overall System Design

The following tables are IPSO charts for the mmltomidi and catmidi programs respectively:

| Inputs | Processes | Storage | Outputs |
|---------------|-----------|--------------------------|-----------------|
| MML text file | MML text | Single-channel MIDI file | Success message |

| Inputs | Processes | Storage | Outputs |
|---------------------------|--------------------|-------------------------|-----------------|
| Single-channel MIDI files | MIDI file contents | Multi-channel MIDI file | Success message |

mmltomidi – User Interface

The mmltomidi program will be called via the terminal with the form shown below:

```
mmltomidi [-o output_path] mml_file
```

The “-o” switch sets the output file to be the “output_path” following. If the switch is not present, then the output file will be called “output.midi” and placed in the working directory. The “mml_file” portion is where the path to the input file is put.

catmidi – User Interface

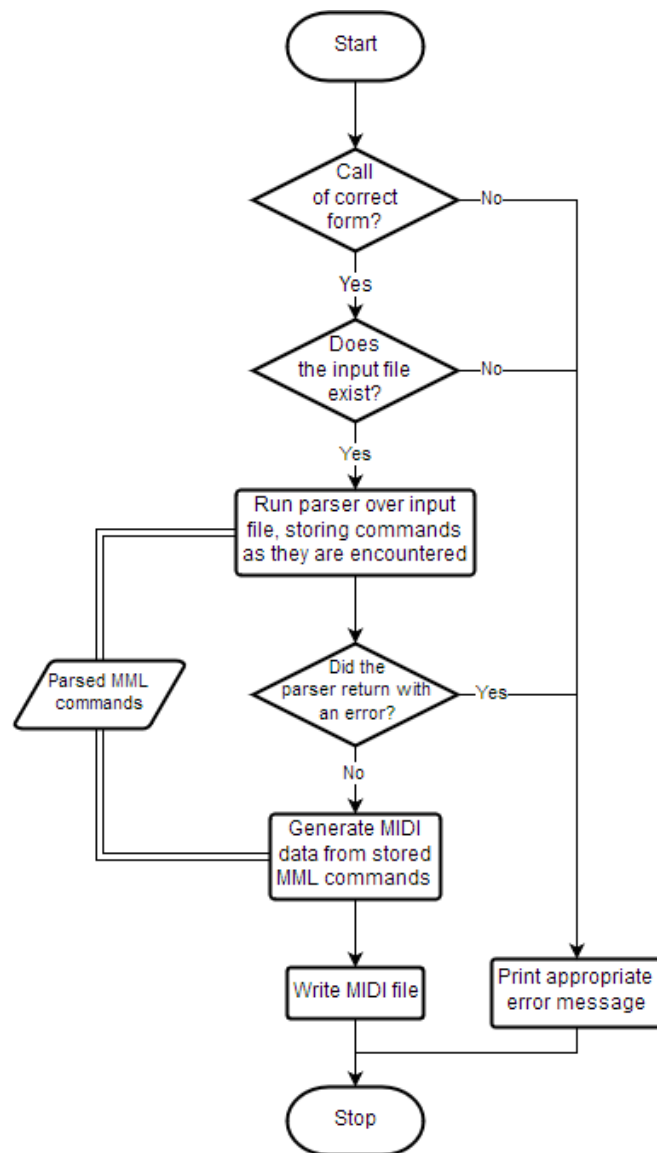
The catmidi program will be called, similarly to mmltomidi, via the terminal with the form shown below:

```
catmidi [-o output_path] [path ...]
```

The “-o” switch will function exactly as described in the mmltomidi user interface section. “[path ...]” is where the paths to the input MIDI files are put, delimited by spaces

mmltomidi – Procedural Abstraction

The following is a flow chart showing a broad abstraction of how the mmltomidi program will work:



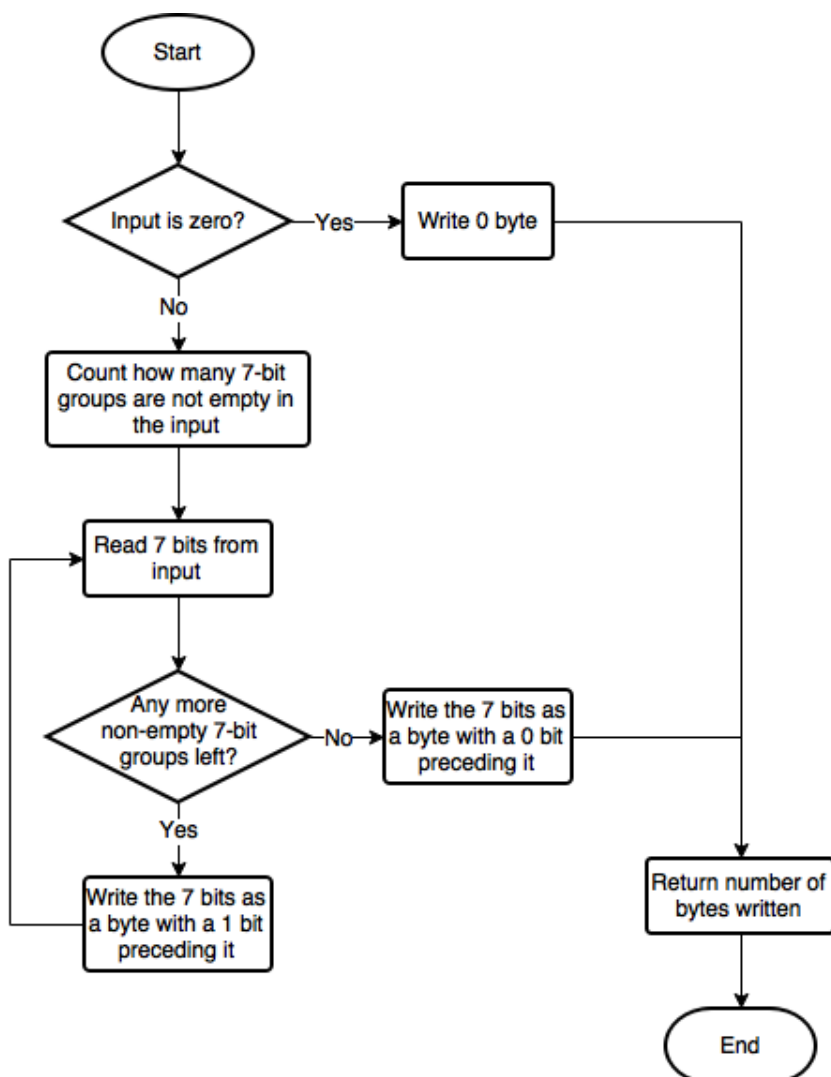
mmltomidi – Key Algorithms

Variable Length Quantity Writing

One of the most utilised algorithms will be to write a variable length quantity. It will take as inputs a pointer to where the data should be written, and the number that should be written as a variable length quantity. It will return the length of the data written. A list of inputs and expected outputs in hexadecimal for this algorithm are shown below.

| Input Number | Data Written | Length of Data Written |
|--------------|--------------|------------------------|
| 00 00 00 00 | 00 | 1 |
| 00 00 00 7F | 7F | 1 |
| 00 00 00 80 | 81 00 | 2 |
| 00 00 3F FF | FF 7F | 2 |
| 00 00 40 00 | 81 80 00 | 3 |
| 0F FF FF FF | FF FF FF 7F | 4 |

A flow chart showing a procedural abstraction of the algorithm is below.



Endianness Swapper

An endianness swapper is another algorithm that will be commonly used throughout the mmltomidi program. It will take a binary number as an input and output the same binary number, but with its endian swapped. The table below shows what outputs would be given for example inputs.

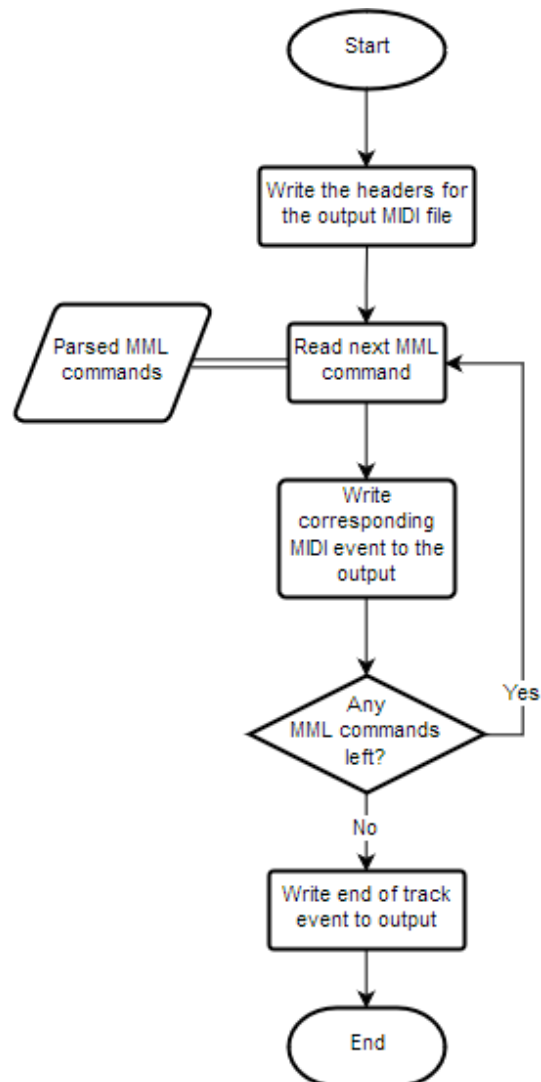
| Input Binary Number | Output Binary Number |
|---------------------|----------------------|
| 00 00 00 00 | 00 00 00 00 |
| 00 00 00 FF | FF 00 00 00 |
| 00 FF 00 00 | 00 00 FF 00 |
| 12 34 56 78 | 78 56 34 12 |

A C-like pseudo-code implementation of this algorithm is shown below.

```
int swapIntEndian(int input) {  
    int output = 0;  
    for each byte in input {  
        write the byte to the byte of opposite significance in output  
    }  
    return output  
}
```

MIDI Data from Processed MML Commands

The key algorithm of the program, however, is that which generates the MIDI data from the processed MML text data. The flowchart below demonstrates how it will function.



mmltomidi – Main Data Structures

The following chunk of code represents the key data structures that store the MML data found by the parser for the MIDI data-generating program.

```
struct note {
    char command; //Letter
    char accidental; //-1 for flat, 1 for
sharp
    unsigned char modifier; //Number after
};

struct mmlFileStruct {
    char name[256]; //Null terminated

    struct note notes[16384];
    int noteCount;
};
```

The main structure is “mmlFileStruct”. This contains the name of the track, the number of notes in the track, and a list of every note and command in the program. The “note” structure is used only to represent a note or command in the “mmlFileStruct” structure. It contains the letter representing the command, whether a note is an accidental or not, and the modifying number for the command.

Below is an example assignment of these data structures.

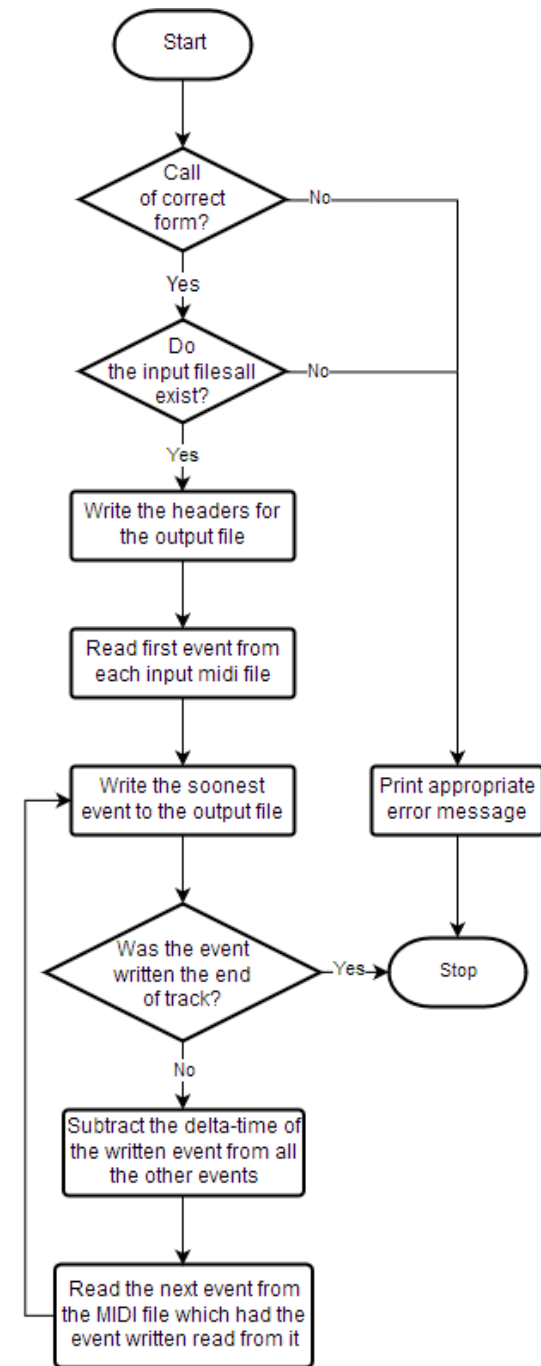
```
struct mmlFileStruct exampleMMLFileStruct;

strcpy(exampleMMLFileStruct.name, “Test track”);
exampleMMLFileStruct.notes[0].command = ‘a’;
exampleMMLFileStruct.notes[0].accidental = 0;
exampleMMLFileStruct.notes[0].modifier = 5;
exampleMMLFileStruct.noteCount = 1
```

This example data represents a MML file with the name “Test track”, which contains a single crochet of note “A”.

catmidi – Procedural Abstraction

The following is a flow chart showing a broad abstraction of how the catmidi program will work:



catmidi – Key Algorithms

The catmidi program shares many of the same algorithms as the mmltomidi program, including the variable length quantity writing algorithm and the endianness swapper. In light of this, only algorithms, which aren't featured in the mmltomidi program, are described in the following sections.

Variable Length Quantity Reading

One of the main algorithms used in the catmidi program is one that reads a variable length quantity and returns an integer. This is very similar to the variable length quantity writing algorithm covered before. It will take a pointer to the variable length quantity to be read as an input, and return an integer. The table below shows what outputs would be expected from the given inputs.

| Input Data | Number Outputted |
|-------------|------------------|
| 00 00 00 00 | 00 |
| 00 00 00 7F | 7F |
| 00 00 81 00 | 80 |
| 00 00 FF 7F | 3F FF |
| 00 81 80 00 | 40 00 |
| FF FF FF 7F | 0F FF FF FF |

An implementation of this algorithm as a C function is shown below.

```
int writeVariableLengthQuantity(char *outputPtr, int input) {
    if (input == 0) {
        *outputPtr = 0;
        return 1;
    }

    int length = 5;

    for (int i = 4; i >= 0; i--) {
        if (input >> i * 7) {
            break;
        } else {
            length--;
        }
    }

    for (int i = length - 1; i >= 0; i--) {
        if (i != 0) {
```



```

        *(outputPtr + length - i - 1) = ((input >> i * 7) & 0x7F) + 0x80;

    } else {
        *(outputPtr + length - i - 1) = (input >> i * 7) & 0x7F;
    }
}

return length;
}

```

Read MIDI Event

A core algorithm used is that which reads an event from an input MIDI file and outputs a structure containing the event and its delta time, with the channel number replaced where appropriate. It will take a pointer to the buffer where the MIDI data is stored, a pointer to an event structure, which is where the event will be stored, and a channel number. The algorithm is broadly demonstrated below in pseudo-code.

```

void readMTrkEvent(unsigned char *inputPtr, struct mtrkEvent *outputPtr, char
channelNumber) {
    outputPtr->deltaTime = readVariableLengthQuantity(inputPtr);

    Copy event to outputPtr->event
    Replace channel number in applicable events
    outputPtr->length = length of event stored
}

```

Combine MIDI Files

The main algorithm of the catmidi program is that which actually combines the MIDI files. It will take a pointer to an output buffer and pointers to MIDI files that have been read into memory. The pseudo-code for this algorithm is shown below.

```

int combineMIDIFiles(char *outputBuffer, unsigned char *inputBuffer[]) {
    write output chunk headers

    Move the input buffer pointers to the start of events
    Read an event from each MIDI file and store them
    while (1) {
        write the event with the smallest delta time to the output

        If this was the end of track MIDI event {
            return length of written MIDI file;
        }

        Subtract delta time of event written from all other event delta times
        Read the next event from the input buffer the event was written from
    }
}

```

catmidi – Main Data Structures

The only data structure in use in the catmidi program is that which stores the read MIDI events. It is shown below as a C structure.

```
struct mtrkEvent {  
    char event[262];  
    short length;  
    int deltaTime;  
};
```

The event itself is stored in “event”, which is sized such that no recognised command will exceed its capacity. The length of the event is stored in “length”. Finally, the delta time of the MIDI event (that is, the time between carrying out the following command and the previous one), is stored in the “deltaTime” integer. In the MIDI file itself this is stored as a variable length quantity.

An example assignment of this structure is shown below, containing an end of track event.

```
struct mtrkEvent exampleEvent;  
  
memcpy(exampleEvent.event, (char []) {0xFF, 0x2F, 0x00}, 3);  
exampleEvent.length = 3;  
exampleEvent.deltaTime = 0;
```

Music Macro Language Design

This section describes the music macro language used by the mmltomidi program.

Introduction

The music macro language (MML) is a music description language that has been in use since 1978, although this was an early version. There has never been an official specification, so each implementation varies slightly, and over the years the language has evolved. The MML to MIDI converter uses a version of the MML derived largely from “Classical MML” with some “Modern MML” features present. Some new specific commands are included also, and some commands are changed where necessary.

All commands in this language have their own line and are terminated by a new line (“\n”, “\r “or “\r\n”).

Comments

Comments are started with two hash characters at the beginning of a new line. This makes the remainder of the line a comment; any more hashes found on the line have no effect. Two hashes are used because single hash starts a meta command.

Playing Notes

The “play” command is used to play a series of notes and macros. Spaces can be intermingled with the notes to improve the clarity of the code. An example usage of this command is shown below:

```
play c5e5g5
```

Note Syntax

Notes are written as the note name followed optionally by the length of the note as a digit – each value for this digit represents a musical note length, which can be seen in the table below. If a length is not given, the default value is used, which is initially 5, but can be changed with the “l” command detailed shortly. A rest is represented by the note name “r”. To play an accidental note a “+” or “-”, respectively, is added after the note name and before the note length. Accidentals applied to rests are ignored.

| MML Note Value Number | Musical Note | |
|--------------------------|-------------------|-------------------|
| | American Notation | Name |
| 0 | 1/32 | Demisemiquaver |
| 1 | 1/16 | Semiquaver |
| 2 | 1/16 + 1/32 | Dotted semiquaver |
| 3 | 1/8 | Quaver |
| 4 | 1/8 + 1/16 | Dotted quaver |
| 5 | 1/4 | Crochet |
| 6 | 1/4 + 1/8 | Dotted crochet |
| 7 | 1/2 | Minim |
| 8 | 1/2 + 1/4 | Dotted minim |
| 9 | 1 | Semibreve |

To alter how each note is played, there are some of commands entered with the notes. These are listed below (where square brackets and their contents are not literal):

- o[digit] Set the octave each following note is played in. The digit represents the scientific pitch notation (SPN) number of the desired octave. All notes entered before this command is entered are played in the 4th SPN octave (“A” will be 440 Hz.)
- < Shift the octave down by one.

- > Shift the octave up by one.
- v[*digit*] Set the volume of the following notes. By default, notes will play at 100% volume.
- p[*number from 0 to 11*] Transpose all the following notes up by the number following 'p' semitones. The default setting is 0.
- l[*digit*] Set the default length of the following notes to the digit. The initial default length is 5. Note that this does not affect the 'v' or 'o' commands.

In modern MML there is also a "t" command, which sets the tempo. This is not included, as a more obvious command on its own line is favoured for ease of reading.

Meta Commands

These commands are entered on their own lines only once and are all preceded by a single hash. They tell the converter how the rest of the file should be played and add information to the MIDI file.

- #tempo [BPM] – set the tempo in BPM of the track (where a beat is a crochet.) This should be set the same in each MML track file when combining them into one MIDI file. The default tempo is 120 BPM.
- #instrument [general MIDI patch number] – set the instrument the rest of the file should be played with. The default instrument is a piano (GM patch number 0.) This command is not present in other MML versions because it is only useful if the file is being converted to a MIDI file.
- #name [name] – set the name of the track. This is put verbatim into the MIDI file in a track name meta event, and can be very useful when altering the MIDI file directly. Only one instance of this command should be in a MML file, otherwise a syntax error will occur.

Macros

A macro in this version of MML is written as below (on its own line):

```
$c v9o4c5
```

The dollar sign shows that this is a macro definition, and the letter following this is the "name" of the macro. The text after the dollar sign and letter replaces any other instance of

the macro name found. A limitation of this notation is that there are only 26 possible macro names, but it is done this way to be more compatible with other versions of MML. Macros can be defined more than once.

Full Example

To conclude this section, a short example of an MML file is shown below.

```
##Example comment

#name test_track
#instrument 0
#tempo 120

$c cdefgab>c

play v8r9$cr9
```

The MIDI file that would be generated from this example is shown below, split into rows in a table with explanations.

| MIDI Data | | Description |
|--|--|---|
| 4D 54 68 64 00 00 00 06 00 00 00 01 00 08 | | Header chunk |
| 4D 54 72 6B 00 00 00 7E | | Track chunk header |
| 00 FF 03 0A 74 65 73 74 5F 74 72 61 63 6B | | Track name assignment |
| 00 FF 58 04 04 02 18 08 | | Time signature assignment |
| 00 FF 51 03 03 D0 90 | | Tempo setting (This is the default tempo) |
| 00 C0 00 | | Patch change (This is the default instrument) |
| 00 C0 00 | | Patch change (This is the instrument specified in the MML file) |
| 00 FF 51 03 03 D0 90 | | Tempo setting (This is the tempo specified in the MML file) |
| 00 90 00 00 20 80 00 00 00 90 3C 70 08 80 3C 70 00 90 3E 70 08 80 3E 70 00 90 40 70 08 80 40 70 00 90 41 70 08 80 41 70 | 00 90 43 70 08 80 43 70 00 90 45 70 08 80 45 70 00 90 47 70 08 80 47 70 00 90 48 70 08 80 48 70 00 90 00 00 20 80 00 00 | This is a string of note on and note off MIDI events that correspond to the notes described in the MML file |
| 00 FF 2F 00 | | |
| | | End of track event |

Finally, below is the sheet music equivalent of the MML example file.



MML in Regex and BNF

This section shows how I will encode the language described in the previous section using regular expressions and Backus-Naur form. These expressions can be put verbatim into the Lex and Yacc input files. The figure below contains the BNF grammar for the MML.

"<mmlFile>" is the start symbol.

```

<mmlFile> ::= <line> | <mmlFile> <line>

<line> ::= <LINE_BREAK> |
          <COMMENT> |
          <TEMPO_SET> |
          <INSTRUMENT_SET> |
          <NAME_SET> |
          <MACRO_ASSIGNED> |
          <PLAY_COMMAND>

```

The following table shows what regexes correspond to what BNF symbol, and gives a description for them.

| Regex | BNF Symbol | Description |
|--|----------------|---|
| <code>^##.*(\r \n (\r\n))</code> | COMMENT | Matches comment lines. |
| <code>^(\r \n (\r\n))</code> | LINE_BREAK | Matches an empty line. |
| <code>^#tempo" "[0-9]{1,3}(\r \n (\r\n))</code> | TEMPO_SET | Matches a tempo setting line. |
| <code>^#instrument" "[0-9]+(\r \n (\r\n))</code> | INSTRUMENT_SET | Matches an instrument assigning line. |
| <code>^#name" "[a-zA-Z0-9_]+(\r \n (\r\n))</code> | NAME_SET | Matches a name setting line. |
| <code>^\$[a-z]" "([cdefgabrov][+]?[0-9]?) (\$[a-z]) (<>) (p[0-9]+) (l[0-9]) " "+(\r \n (\r\n))</code> | MACRO_ASSIGNED | Matches a macro assignment. The middle portion is the regex that matches series of notes. |
| <code>^play" "([cdefgabrov][+]?[0-9]?) (\$[a-z]) (<>) (p[0-9]{1,2}) (l[0-9]) " "+(\r \n (\r\n))</code> | PLAY_COMMAND | Matches a play command. The middle portion is, again, the regex that matches a series of notes. |

Technical Solution

mmltomidi/makefile

```

BUILD_DIR = ./build
STD = -std=c99
OPTI = -O3
EXE = mmltomidi

$(EXE): main.c main.h y.tab.c lex.yy.c
    gcc -g -w $(STD) $(OPTI) -o $(BUILD_DIR)/$(EXE) main.c $(BUILD_DIR)/lex.yy.c
$(BUILD_DIR)/y.tab.c

lex.yy.c: lex.l y.tab.h
    lex -s -o $(BUILD_DIR)/lex.yy.c lex.l

y.tab.c y.tab.h: yacc.y
    yacc -d -o $(BUILD_DIR)/y.tab.c yacc.y

.PHONY: clean

clean:
    rm -f $(BUILD_DIR)/$(EXE)
    rm -f $(BUILD_DIR)/*.o
    rm -f $(BUILD_DIR)/*.c
    rm -f $(BUILD_DIR)/*.h

```

mmltomidi/main.c

```

#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <unistd.h>

#include "main.h"
#include "mmlFileStruct.h"

struct mmlFileStruct processedMmlFile; //Necessary global for info from lex.yy
extern FILE *yyin; //For linking to lex.yy.c
extern bool macroEnabled[26]; //Necessary for clearing

int swapIntEndianness(int input) {
    //Swaps integer endianness

    int output = 0;

    for (int i = 0; i < 4; i++) {
        *((char *) &output - i + 3) = *((char *) &input + i);
    }

    return output;
}

int swapShortEndianness(short input) {
    //Swaps short endianness

    int output = 0;

    for (int i = 0; i < 2; i++) {
        *((char *) &output - i + 1) = *((char *) &input + i);
    }

    return output;
}

int writevariableLengthQuantity(char *outputPtr, int input) {
    //Writes "input" as a variable length quantity at where "outputPtr" points to.
    //Returns length of variable length quantity written

    if (input == 0) {
        *outputPtr = 0;
        return 1;
    }

```

```

}

int length = 5;
for (int i = 4; i >= 0; i--) {
    if (input >> i * 7) {
        break;
    } else {
        length--;
    }
}

for (int i = length - 1; i >= 0; i--) {
    if (i != 0) {
        *(outputPtr + length - i - 1) = ((input >> i * 7) & 0x7F) + 0x80;
    } else {
        *(outputPtr + length - i - 1) = (input >> i * 7) & 0x7F;
    }
}

return length;
}

void writeMTrkHeader(struct mtrkHeader *mtrkHeaderPtr, int trackLength) {
    //Fills in a given mtrkHeader struct

    strncpy(mtrkHeaderPtr->chunkType, "MTrk", 4);
    mtrkHeaderPtr->length = swapIntEndianness(trackLength);
}

void writeMThdHeader(struct mthdHeader *mthdHeaderPtr) {
    //Fills in a given mthdHeader struct

    strncpy(mthdHeaderPtr->chunkType, "MThd", 4);
    mthdHeaderPtr->length = swapIntEndianness(6);
    mthdHeaderPtr->format = 0;
    mthdHeaderPtr->ntrks = swapShortEndianness(1);
    mthdHeaderPtr->division = swapShortEndianness(8);
}

int generateMIDIFile(char **outputPtr, struct mmfFileStruct *midiData) {
    //Points outputPtr towards a malloc assigned mmf array containing the MIDI
    //file described by midiData

    *outputPtr = malloc(65536);

    if (*outputPtr == NULL) {
        fprintf(stderr, "Error - memory could not be assigned by malloc\n");
        return NULL;
    }

    struct mthdHeader *outputMThdHeader = *outputPtr;
    struct mtrkHeader *outputMTrkHeader = *outputPtr + 14;
    char *trackChunkPtr = *outputPtr + 22;

    writeMThdHeader(outputMThdHeader);

    if (midiData->name[0]) {
        memcpy(trackChunkPtr, (char []) {0x00, 0xff, 0x03, strlen(midiData->name)}, 4);
        strcpy(trackChunkPtr += 4, midiData->name);
        trackChunkPtr += strlen(midiData->name);
    }

    //Default time signature
    memcpy(trackChunkPtr, (char []) {0x00, 0xFF, 0x58, 0x04, 0x04, 0x02, 0x18, 0x08},
8);
    trackChunkPtr += 8;

    //Default tempo
    memcpy(trackChunkPtr, (char []) {0x00, 0xFF, 0x51, 0x03}, 4);
    *((int *) (trackChunkPtr += 4)) = swapIntEndianness(3000000 / 120) >> 8;
    trackChunkPtr += 3;

    //Default instrument
    memcpy(trackChunkPtr, (char []) {0x00, 0xc0, 0x00}, 3);

```



```

trackChunkPtr += 3;

char octave = 4;
char velocity = 0x7F;
char transposition = 0;
char noteLookup[7] = {21, 23, 12, 14, 16, 17, 19};
char deltaTimeLookup[10] = {1, 2, 3, 4, 6, 8, 12, 16, 24, 32};

for (int i = 0; i < midiData->noteCount; i++) {
    struct note currentNote = midiData->notes[i];

    switch (currentNote.command) {
        case 'o':
            octave = currentNote.modifier;
            break;

        case '<':
            octave -= octave != 0;
            break;

        case '>':
            octave += octave != 9;
            break;

        case 'p':
            transposition = currentNote.modifier;
            break;

        case 'v':
            velocity = (0x7F * currentNote.modifier) / 9;
            break;

        case 't':
            memcpy(trackChunkPtr, (char []) {0x00, 0xFF, 0x51, 0x03}, 4);
            *((int *) (trackChunkPtr += 4)) = swapIntEndianness(30000000 /
currentNote.modifier) >> 8;
            trackChunkPtr += 3;

            break;

        case 'i':
            memcpy(trackChunkPtr, (char []) {0x00, 0xC0, currentNote.modifier}, 3);
            trackChunkPtr += 3;

            break;

        default:
            ; //Should always be a note

            char noteNumber = noteLookup[currentNote.command - 'a'] + 12 * octave +
currentNote.accidental + transposition;

            memcpy(trackChunkPtr, (char []) {0x00, 0x90, (currentNote.command != 'r') *
noteNumber, (currentNote.command != 'r') * velocity}, 4);
            trackChunkPtr += 4;

            trackChunkPtr += writeVariableLengthQuantity(trackChunkPtr,
deltaTimeLookup[currentNote.modifier]);
            memcpy(trackChunkPtr, (char []) {0x80, (currentNote.command != 'r') *
noteNumber, (currentNote.command != 'r') * velocity}, 3);
            trackChunkPtr += 3;

            break;
    }
}

//End of track message
memcpy(trackChunkPtr, (char []) {0x00, 0xFF, 0x2F, 0x00}, 4);
trackChunkPtr += 4;

writeMTrkHeader(outputMTrkHeader, trackChunkPtr - *outputPtr - 22);

*outputPtr = realloc(*outputPtr, trackChunkPtr - *outputPtr + 1);

if (*outputPtr == NULL) {
    fprintf(stderr, "Error - malloc'd array could not be reallocated\n");
}

```

```

    return NULL;
}

return trackChunkPtr - *outputPtr;
}

bool fileReadable(char *path) {
    //Checks if the program has the rights to read the input file

    if (access(path, R_OK)) {
        fprintf(stderr, "File %s is not readable\n", path);
        return false;
    }

    return true;
}

bool pathValid(char *path) {
    //Checks if the input path leads to something
    //Currently does not error on a directory

    if (access(path, F_OK)) {
        fprintf(stderr, "File %s does not exist\n", path);
        return false;
    }

    return true;
}

bool correctCallForm(int argc, char *argv[]) {
    //Returns a boolean for if the call is of the right form

    if ((argc != 2) && (argc != 4)) {
        fprintf(stderr, "Invalid number of arguments given\n");
        return false;
    }

    return true;
}

int main(int argc, char *argv[]) {
    if (!correctCallForm(argc, argv)) {
        fprintf(stderr, "Usage: mmltomidi [-o output_path] file\n");
        return 1;
    }

    bool outputPathGiven = (strcmp(argv[1], "-o")) ? false : true;
    char *outputPath = (outputPathGiven) ? argv[2] : "output.midi";
    char *inputPath = (outputPathGiven) ? argv[3] : argv[1];

    if ((!pathValid(inputPath)) || (!fileReadable(inputPath))) {
        return 1;
    }

    memset(&processedMmlFile, 0, sizeof(processedMmlFile));
    memset(&macroEnabled, 0, 26);

    yyin = fopen(inputPath, "rb");
    int yyparseResult = yyparse();
    fclose(yyin);

    if (yyparseResult == 1) {
        fprintf(stderr, "Syntax error encountered by parser - terminating\n");
        return 1;
    }

    char *midiBuffer;
    int midiBufferLength = generateMIDIFile(&midiBuffer, &processedMmlFile);

    if (midiBuffer == NULL) {
        return 1;
    }

    FILE *outputFile = fopen(outputPath, "wb");

```

```

    if (outputFile == NULL) {
        fprintf(stderr, "Output file could not be created/opened\n");
        return 1;
    }

    fwrite(midiBuffer, 1, midiBufferLength, outputFile);

    free(midiBuffer);
    fclose(outputFile);

    return 0;
}

```

mmltomidi/main.h

```

#ifndef MAIN_H
#define MAIN_H

extern int yyparse (void); //Function prototype for linking to y.tab.c

struct mthdHeader {
    char chunkType[4];
    int length;
    short format;
    short ntrks;
    short division;
};

struct mtrkHeader {
    char chunkType[4];
    int length;
};

#endif

```

mmltomidi/lex.l

```

%{
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#include "y.tab.h"
#include "../mmlFileStruct.h"

extern struct mmlFileStruct processedMmlFile;

char macroTable[26][256]; //Null terminated
bool macroEnabled[26];

void replaceSubstring(char *source, char *find, char *replace) {
    //Replaces the substring "find" with "replace" in "source"
    //Requires source to point to a malloc assigned array

    int matches = 0;
    char *i = strstr(source, find);

    while (i != NULL) {
        matches++;
        i = strstr(i + 1, find);
    }

    if (!matches) {
        return;
    }

    char originalSource[strlen(source)];
    strcpy(originalSource, source);

    source = realloc(source, strlen(source) - matches * strlen(find) + matches *

```

```

strlen(replace));

    int sourceLen = strlen(source) - matches * strlen(find) + matches *
strlen(replace);

    if (source == NULL) {
        return;
    }

    int oldIndex = 0, newIndex = 0;
    char *nextMatch = NULL;

    while (oldIndex != strlen(originalSource)) {
        if (nextMatch == NULL) {
            nextMatch = strstr(originalSource + oldIndex, find);

            if (nextMatch == NULL) {
                strcpy(source + newIndex, originalSource + oldIndex);
                break;
            }
        }

        if (nextMatch == originalSource + oldIndex) {
            strcpy(source + newIndex, replace);
            newIndex += strlen(replace);
            oldIndex += strlen(find);
            nextMatch = NULL;
        } else {
            source[newIndex++] = originalSource[oldIndex++];
        }
    }
}

%}

%%
^##.*(\r|\n|(\r\n)) {
    return COMMENT;
}

^(\r|\n|(\r\n)) {
    return LINE_BREAK;
}

^#tempo" "[0-9]{1,3}(\r|\n|(\r\n)) {
    int tempoNumber = atoi(yytext + 7);

    if (tempoNumber > 255) {
        strcpy(yyval.errorMessage, "Error - tempo set higher than 255");
        return ERROR;
    }

    struct note *currentNote = processedMmlFile.notes + processedMmlFile.noteCount;
    currentNote->command = 't';
    currentNote->modifier = tempoNumber;
    processedMmlFile.noteCount++;

    return TEMPO_SET;
}

^#instrument" "[0-9]+(\r|\n|(\r\n)) {
    int instrumentNumber = atoi(yytext + 11);

    if (instrumentNumber > 127) {
        strcpy(yyval.errorMessage, "Error - instrument set higher than 127");
        return ERROR;
    }

    struct note *currentNote = processedMmlFile.notes + processedMmlFile.noteCount;
    currentNote->command = 'i';
    currentNote->modifier = instrumentNumber;
    processedMmlFile.noteCount++;

    return INSTRUMENT_SET;
}

```

```

^#name" "[a-zA-Z0-9_]+(\r|\n|(\r\n)) {
    static bool nameSet = false;

    if (nameSet) {
        strcpy(yylval.errorMessage, "Error - name set more than once");
        return ERROR;
    }

    nameSet = true;
    int i = 5;

    while (yytext[i] != '\n' && yytext[i] != '\r') {
        if (i == 261) {
            strcpy(yylval.errorMessage, "Error - name set was too long: limit is 255");
            return ERROR;
        }

        processedMmlFile.name[i - 6] = yytext[i++];
    }

    processedMmlFile.name[i - 6] = '\0';

    return NAME_SET;
}

^$[a-z]" "(((cdefgabrov)[+-]?[0-9]?)|($[a-z])|(<>)|(p[0-9]+)|(l[0-9])|"
")+(\r|\n|(\r\n)) {
    int i = 3;

    while (yytext[i] != '\n' && yytext[i] != '\r') {
        if (i == 258) {
            strcpy(yylval.errorMessage, "Macro assigned was too long - limit is 255");
            return ERROR;
        }

        macroTable[yytext[1] - 'a'][i - 3] = yytext[i++];
    }

    macroEnabled[yytext[1] - 'a'] = true;
    macroTable[yytext[1] - 'a'][i - 3] = '\0';

    return MACRO_ASSIGNED;
}

^play" "(((cdefgabrov)[+-]?[0-9]?)|($[a-z])|(<>)|(p[0-9]{1,2})|(l[0-9])|"
")+(\r|\n|(\r\n)) {
    char *processedYYText = malloc(strlen(yytext)); //Malloc required for
    replaceSubstring

    if (processedYYText == NULL) {
        strcpy(yylval.errorMessage, "Error - memory could not be malloc'd");
        return ERROR;
    }

    strcpy(processedYYText, yytext);

    for (int i = 0; i < strlen(yytext); i++) {
        if ((yytext[i] == '$') && (!macroEnabled[yytext[i + 1] - 'a'])) {
            strcpy(yylval.errorMessage, "Macro used but not assigned");
            return ERROR;
        }
    }

    for (char i = 0; i < 26; i++) {
        if (macroEnabled[i]) {
            replaceSubstring(processedYYText, (char []) {'$', i + 'a', 0x00},
            macroTable[i]);

            if (processedYYText == NULL) {
                strcpy(yylval.errorMessage, "Error - memory assigned by malloc could not be
                reallocated");
                return ERROR;
            }
        }
    }

    int index = 5;

```

```

char defaultLength = 5;
struct note *currentNote;

while (index < strlen(processedYYText)) {
    currentNote = processedMmlFile.notes + processedMmlFile.noteCount;

    if (strchr("cdefgabrov", processedYYText[index]) != NULL) {
        currentNote->command = processedYYText[index++];
        currentNote->accidental = 0;
        currentNote->modifier = defaultLength;

        while (1) {
            if ((index < strlen(processedYYText)) && (strchr("+-",
processedYYText[index]) != NULL)) {
                currentNote->accidental = (processedYYText[index++] == '+') ? 1 : -1;
                continue;
            }

            if ((index < strlen(processedYYText)) && (strchr("0123456789",
processedYYText[index]) != NULL)) {
                currentNote->modifier = processedYYText[index++] - '0';
                continue;
            }

            break;
        }

        processedMmlFile.noteCount++;

    } else if (strchr("<>", processedYYText[index]) != NULL) {
        currentNote->command = processedYYText[index++];
        processedMmlFile.noteCount++;

    } else if (processedYYText[index] == 'p') {
        currentNote->command = processedYYText[index++];
        currentNote->modifier = 0;

        if ((index + 1 < strlen(processedYYText)) && (strchr("0123456789",
processedYYText[index + 1]) != NULL)) {
            currentNote->modifier = (processedYYText[index++] - '0') * 10;
        }

        currentNote->modifier += processedYYText[index++] - '0';

        if (currentNote->modifier > 11) {
            strcpy(yylval.errorMessage, "Transposition set too high - only values from
0 to 11 are valid");
            return ERROR;
        }

        processedMmlFile.noteCount++;

    } else if (processedYYText[index] == 'l') {
        defaultLength = processedYYText[++index] - '0';
        index++;

    } else {
        //Should never occur, as such an input should not get past the regex
        index++;
    }
}

free(processedYYText);

return PLAY_COMMAND;
}

.\n\r {
    return ERROR;
}
%%

```

mmltomidi/mmlFileStruct.h

```

#ifndef MMLFILESTRUCT_H
#define MMLFILESTRUCT_H

//Necessary for only one copy of the struct definition to exist

struct note {
    char command;
    char accidental; //-1 for flat, 1 for sharp
    unsigned char modifier;
};

struct mmlFileStruct {
    char name[256]; //Null terminated

    struct note notes[16384];
    int noteCount;
};

#endif

```

mmltomidi/yacc.y

```

%{
#include <stdio.h>
#include <stdbool.h>

void yyerror(char *s) {
    fprintf(stderr, "%s\n", s);
}

int yywrap() {
    return 1;
}

%}

%union {
    int value;
    char errorMessage[256];
}

%start mmlFile

%token <value> COMMENT
%token <value> LINE_BREAK
%token <value> TEMPO_SET
%token <value> INSTRUMENT_SET
%token <value> NAME_SET
%token <value> MACRO_ASSIGNED
%token <value> PLAY_COMMAND
%token <errorMessage> ERROR

%type <value> line

%%
mmlFile: line
| mmlFile line
;

line: LINE_BREAK
| COMMENT
| TEMPO_SET
| INSTRUMENT_SET
| NAME_SET
| MACRO_ASSIGNED
| PLAY_COMMAND
| ERROR {
    fprintf(stderr, "%s\n", $1);
    YYERROR;
};
%%

```

catmidi/main.c

```

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h>

#include "main.h"

int readVariableLengthQuantity(char *inputPtr) {
    //Reads a variable length quantity from "inputPtr" and returns it as an integer

    char *workingPtr = inputPtr;

    while (*workingPtr & 0x80) {
        workingPtr++;
    }

    int output = 0;
    int outputShift = 0;

    do {
        output |= (*workingPtr & 0x7F) << outputShift;
        outputShift += 7;
    } while (workingPtr-- != inputPtr);

    return output;
}

int writeVariableLengthQuantity(char *outputPtr, int input) {
    //writes "input" as a variable length quantity at where "outputPtr" points to.
    //Returns length of variable length quantity written

    if (input == 0) {
        *outputPtr = 0;
        return 1;
    }

    int length = 5;

    for (int i = 4; i >= 0; i--) {
        if (input >> i * 7) {
            break;
        }
        else {
            length--;
        }
    }

    for (int i = length - 1; i >= 0; i--) {
        if (i != 0) {
            *(outputPtr + length - i - 1) = ((input >> i * 7) & 0x7F) + 0x80;
        }
        else {
            *(outputPtr + length - i - 1) = (input >> i * 7) & 0x7F;
        }
    }

    return length;
}

char readMTrkEvent(unsigned char **inputPP, struct mtrkEvent *outputPtr, char
channelNumber) {
    //Reads a MIDI track event from the double pointer "inputPP"
    //Increases the pointer as the file is read through
    //Stores results in "outputPtr"
    //Returns non-zero on error

    outputPtr->deltaTime = readVariableLengthQuantity((char *) *inputPP);

    while (**inputPP & 0x80) {
        (*inputPP)++;
    }
}

```



```

}

(*inputPP)++;
unsigned char *originalPtr = *inputPP;
switch (**inputPP) {
    case 0xFF:
        (*inputPP)++;

        switch (**inputPP) {
            case 0x03: //Name
                (*inputPP)++;
                *inputPP += **inputPP + 1;

                break;

            case 0x2f: //End
                *inputPP += 2;

                break;

            case 0x51: //Tempo
                *inputPP += 5;

                break;

            case 0x58: //Time sig.
                *inputPP += 6;

                break;

            default:
                fprintf(stderr, "Unknown MTrk event encountered\n");

                return 1;
        }

        break;

    case 0x80: //Note off
    case 0x90: //Note on
        **inputPP |= channelNumber;
        *inputPP += 3;

        break;

    case 0xC0: //Patch change
        **inputPP |= channelNumber;
        *inputPP += 2;

        break;

    default:
        fprintf(stderr, "Unknown MTrk event encountered\n");

        return 1;
}

outputPtr->length = *inputPP - originalPtr;
memcpy(outputPtr->event, originalPtr, outputPtr->length);

return 0;
}

int swapIntEndianness(int input) {
    //Swaps integer endianness

    int output = 0;

    for (int i = 0; i < sizeof(int); i++) {
        *((char *) &output + sizeof(int) - i - 1) = *((char *) &input + i);
    }

    return output;
}

```

```

int swapShortEndianness(short input) {
    //Swaps short endianness

    int output = 0;

    for (int i = 0; i < sizeof(short); i++) {
        *((char *) &output + sizeof(short) - i - 1) = *((char *) &input + i);
    }

    return output;
}

void writeMTrkHeader(struct mtrkHeader *mtrkHeaderPtr, int trackLength) {
    //Fills in a given mtrkHeader struct

    strncpy(mtrkHeaderPtr->chunkType, "MTrk", 4);
    mtrkHeaderPtr->length = swapIntEndianness(trackLength);
}

void writeMThdHeader(struct mthdHeader *mthdHeaderPtr) {
    //Fills in a given mthdHeader struct

    strncpy(mthdHeaderPtr->chunkType, "MThd", 4);
    mthdHeaderPtr->length = swapIntEndianness(6);
    mthdHeaderPtr->format = 0;
    mthdHeaderPtr->ntrks = swapShortEndianness(1);
    mthdHeaderPtr->division = swapShortEndianness(8);
}

int combineMIDIFiles(char *outputBuffer, unsigned char *inputBuffer[], char
inputBufferCount) {
    //Combines raw MIDI files passed with the "inputBuffer" array
    //Outputs the combined file to "outputBuffer"
    //Returns zero on error

    struct mthdHeader *outputMThdHeader = (void *) outputBuffer;
    struct mtrkHeader *outputMTrkHeader = (void *) outputBuffer + 14;
    char *trackPtr = (void *) outputBuffer + 22;

    writeMThdHeader(outputMThdHeader);

    unsigned char *inputBufferPtr[inputBufferCount];
    struct mtrkEvent inputEvent[inputBufferCount];

    for (int i = 0; i < inputBufferCount; i++) {
        inputBufferPtr[i] = inputBuffer[i] + 22;

        if (readMTrkEvent(&inputBufferPtr[i], &inputEvent[i], i)) {
            return 0;
        }
    }

    bool nameSet = false;
    bool timeSignatureSet = false;

    int smallestDeltaTime;
    char soonestEventIndex;
    struct mtrkEvent *soonestEvent;

    while (1) {
        smallestDeltaTime = inputEvent[0].deltaTime;
        soonestEventIndex = 0;

        for (int i = 1; i < inputBufferCount; i++) {
            if (inputEvent[i].deltaTime < smallestDeltaTime) {
                smallestDeltaTime = inputEvent[i].deltaTime;
                soonestEventIndex = i;
            }
        }

        for (int i = 0; i < inputBufferCount; i++) {
            if (i == soonestEventIndex) {
                continue;
            }

            inputEvent[i].deltaTime -= smallestDeltaTime;
        }
    }
}

```

```

    soonestEvent = &inputEvent[soonestEventIndex];

    if (!memcmp(soonestEvent->event, (char []) {0xFF, 0x03}, 2)) { //Name setting
event
        if (nameSet) {
            if (readMTrkEvent(&inputBufferPtr[soonestEventIndex],
&inputEvent[soonestEventIndex], soonestEventIndex)) {
                return 0;
            }
            continue;
        }
        nameSet = true;
    }

    if (!memcmp(soonestEvent->event, (char []) {0xFF, 0x58, 0x04}, 3)) { //Time
signature setting event
        if (timeSignatureSet) {
            if (readMTrkEvent(&inputBufferPtr[soonestEventIndex],
&inputEvent[soonestEventIndex], soonestEventIndex)) {
                return 0;
            }
            continue;
        }
        timeSignatureSet = true;
    }

    trackPtr += writeVariableLengthQuantity(trackPtr, soonestEvent->deltaTime);
    memcpy(trackPtr, soonestEvent->event, soonestEvent->length);
    trackPtr += soonestEvent->length;

    if (memcmp(soonestEvent->event, (char []) {0xFF, 0x2F, 0x00}, 3) == 0) { //End
of track event
        break;
    }

    if (readMTrkEvent(&inputBufferPtr[soonestEventIndex],
&inputEvent[soonestEventIndex], soonestEventIndex)) {
        return 0;
    }
}

writeMTrkHeader(outputMTrkHeader, trackPtr - outputBuffer - 22);
return trackPtr - outputBuffer;
}

int loadFile(char *path, unsigned char **buffer) {
    //Loads a file stored at "path"

    FILE *f = fopen(path, "rb");

    fseek(f, 0, SEEK_END);
    int length = ftell(f);
    rewind(f);

    *buffer = malloc(length);
    fread(*buffer, 1, length, f);

    fclose(f);

    return length;
}

bool fileReadable(char *path) {
    //Checks if the program has the rights to read the input file

    if (access(path, R_OK)) {
        fprintf(stderr, "File %s is not readable\n", path);
        return false;
    }

    return true;
}

```

```

}

bool pathValid(char *path) {
    //Checks if the input path leads to something
    //Currently does not error on a directory

    if (access(path, F_OK)) {
        fprintf(stderr, "File %s does not exist\n", path);
        return false;
    }

    return true;
}

bool correctCallForm(int argc, char *argv[]) {
    //Returns a boolean for if the call is of the right form

    if (argc < 2) {
        fprintf(stderr, "Too few arguments supplied\n");
        return false;
    }

    if (argc > 19) {
        fprintf(stderr, "Too many arguments supplied - 16 files can be combined at most\n");
        return false;
    }

    if ((!strcmp(argv[1], "-o")) && (argc < 4)) {
        return false;
    }

    return true;
}

int main(int argc, char *argv[]) {
    if (!correctCallForm(argc, argv)) {
        fprintf(stderr, "Usage: catmidi [-o output_path] [path ...]\n");
        return 1;
    }

    bool outputPathGiven = !strcmp(argv[1], "-o");
    char startOfInputs = (outputPathGiven) ? 3 : 1;
    char numberOfInputs = argc - startOfInputs;

    for (int i = startOfInputs; i < argc; i++) {
        if ((!pathValid(argv[i])) || (!fileReadable(argv[i]))) {
            return 1;
        }
    }

    char outputBuffer[65536];
    unsigned char *inputBuffer[numberOfInputs];

    for (int i = 0; i < numberOfInputs; i++) {
        loadFile(argv[i + startOfInputs], &inputBuffer[i]);
    }

    int outputBufferLength = combineMIDIFiles(outputBuffer, inputBuffer,
        numberOfInputs);

    for (int i = 0; i < numberOfInputs; i++) {
        free(inputBuffer[i]);
    }

    if (outputBufferLength == 0) {
        return 1;
    }

    FILE *outputFile = fopen((outputPathGiven) ? argv[2] : "./output.midi", "wb");

    if (outputFile == NULL) {
        fprintf(stderr, "Output file could not be created\n");
    }
}

```

```
    return 1;
}

fwrite(outputBuffer, 1, outputBufferLength, outputFile);
fflush(outputFile);
fclose(outputFile);

return 0;
}
```

catmidi/main.h

```
#ifndef MAIN_H
#define MAIN_H

struct mthdHeader {
    char chunkType[4];
    int length;
    short format;
    short ntrks;
    short division;
};

struct mtrkHeader {
    char chunkType[4];
    int length;
};

struct mtrkEvent {
    char event[259];
    short length;
    int deltaTime;
};

#endif
```

catmidi/makefile

```
BUILD_DIR = ./build
STD = -std=c99
OPTI = -O3
EXE = catmidi

$(EXE): main.o
    gcc -g -w $(STD) $(OPTI) -o $(BUILD_DIR)/$(EXE) $(BUILD_DIR)/main.o

main.o: main.c
    gcc -c $(STD) $(OPTI) -o $(BUILD_DIR)/main.o main.c

.PHONY: clean

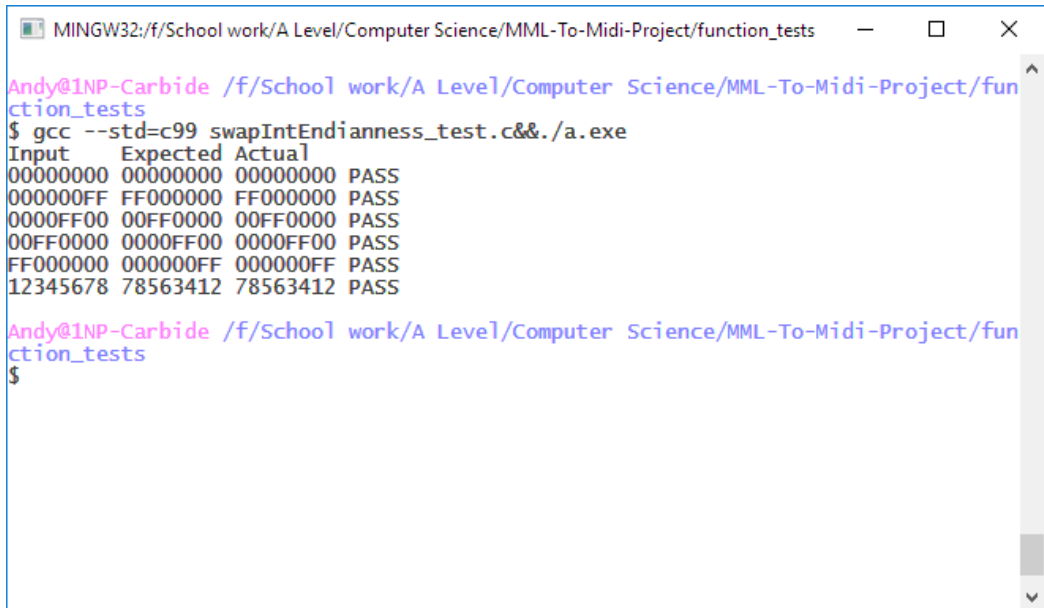
clean:
    rm -f $(BUILD_DIR)/$(EXE)
    rm -f $(BUILD_DIR)/*.o
```

Testing

When I started the project, I planned to use unit testing, and perform test driven development; due to the project making heavy use of the compiler compilers lex and yacc, however, I could not come up with a good way of performing unit testing on my programs. In light of this, I've decided to carry out testing of key functions in isolated source files with tailored test cases.

Function Tests

| | |
|--------------------|---|
| Function prototype | <code>int swapIntEndianness(int input)</code> |
| Purpose | Swap the endianness of the integer "input" and return it. |
| Testing code | <pre> #include <stdio.h> struct testCase { int input; int expectedOutput; }; int swapIntEndianness(int input) { int output = 0; for (int i = 0; i < 4; i++) { *((char *) &output - i + 3) = *((char *) &input + i); } return output; } int main(int argc, char *argv[]) { struct testCase tests[] = {{0x00000000, 0x00000000}, {0x000000FF, 0xFF000000}, {0x0000FF00, 0x00FF0000}, {0x00FF0000, 0x0000FF00}, {0xFF000000, 0x000000FF}, {0x12345678, 0x78563412}}; int actualOutput; printf("Input Expected Actual \n"); for (int i = 0; i < sizeof(tests) / sizeof(struct testCase); i++) { actualOutput = swapIntEndianness(tests[i].input); printf("%08x %08x %08x", tests[i].input, tests[i].expectedOutput, actualOutput); if (actualOutput != tests[i].expectedOutput) { printf(" FAIL\n"); continue; } printf(" PASS\n"); } return 0; } </pre> |

| | |
|---------|--|
| Outcome |  <pre> MINGW32:/f/School work/A Level/Computer Science/MML-To-Midi-Project/function_tests Andy@1NP-Carbid /f/School work/A Level/Computer Science/MML-To-Midi-Project/function_tests \$ gcc --std=c99 swapIntEndianness_test.c&&./a.exe Input Expected Actual 00000000 00000000 00000000 PASS 000000FF FF000000 FF000000 PASS 0000FF00 00FF0000 00FF0000 PASS 00FF0000 0000FF00 0000FF00 PASS FF000000 000000FF 000000FF PASS 12345678 78563412 78563412 PASS Andy@1NP-Carbid /f/School work/A Level/Computer Science/MML-To-Midi-Project/function_tests \$ </pre> |
|---------|--|

| | |
|--------------------|---|
| Function prototype | <code>int writeVariableLengthQuantity(char *outputPtr, int input)</code> |
| Purpose | write the integer "input" as a big endian variable length quantity at the pointer "outputPtr". Return the length of the variable length quantity written. |
| Testing code | <pre> #include <stdio.h> struct testCase { int input; int expectedVLQ; int expectedReturn; }; int writeVariableLengthQuantity(char *outputPtr, int input) { if (input == 0) { *outputPtr = 0; return 1; } int length = 5; for (int i = 4; i >= 0; i--) { if (input >> i * 7) { break; } else { length--; } } for (int i = length - 1; i >= 0; i--) { if (i != 0) { *(outputPtr + length - i - 1) = ((input >> i * 7) & 0x7F) + 0x80; } else { *(outputPtr + length - i - 1) = (input >> i * 7) & 0x7F; } } return length; } </pre> |

```

int main(int argc, char *argv[]) {
    struct testCase tests[] = {
        {0x00000000, 0x00000000, 0},
        {0x00000040, 0x00000040, 1},
        {0x0000007F, 0x0000007F, 1},
        {0x00000080, 0x00000081, 2},
        {0x00002000, 0x000000C0, 2},
        {0x00003FFF, 0x00007FFF, 2},
        {0x00004000, 0x00008081, 3},
        {0x00100000, 0x000080C0, 3},
        {0x001FFFFFF, 0x007FFFFFF, 3},
        {0x00200000, 0x00808081, 4},
        {0x08000000, 0x008080C0, 4},
        {0xFFFFFFFF, 0x7FFFFFFF, 4}};

    int actualVLQ;
    int actualReturn;

    printf("Input      Expected Expected Actual   Actual   \n");
    printf("      VLQ      Return   VLQ      Return   \n");

    for (int i = 0; i < sizeof(tests) / sizeof(struct testCase); i++) {
        actualVLQ = 0;
        actualReturn = writeVariableLengthQuantity((char *) &actualVLQ,
        tests[i].input);

        printf("%08x %08x %08x %08x %08x", tests[i].input,
        tests[i].expectedVLQ, tests[i].expectedReturn, actualVLQ, actualReturn);

        if (actualVLQ != tests[i].expectedVLQ) {
            printf(" FAIL\n");
            continue;
        }

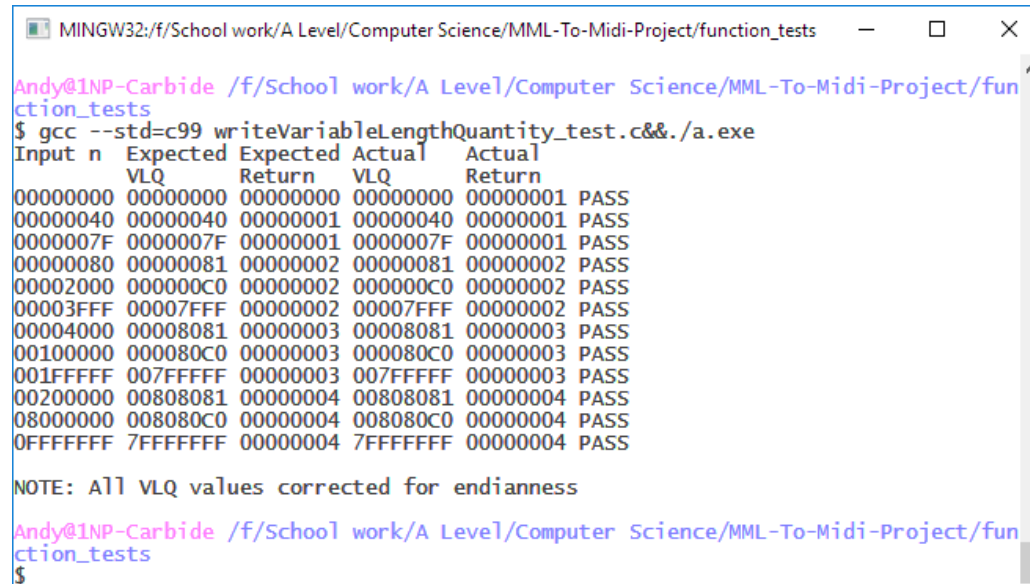
        printf(" PASS\n");
    }

    printf("\nNOTE: All VLQ values corrected for endianness\n");

    return 0;
}

```

Outcome



```

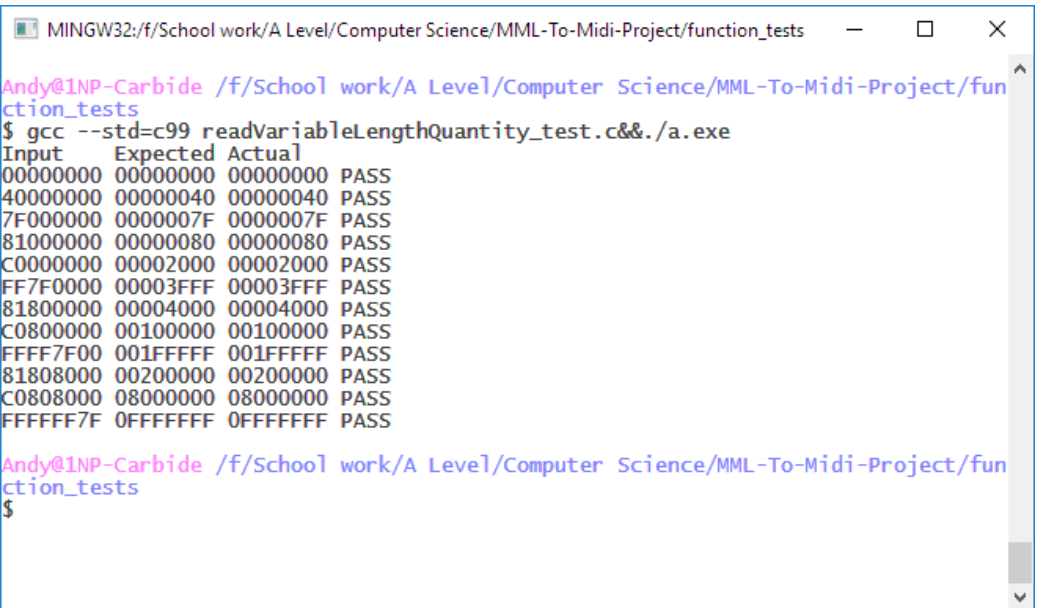
MINGW32:/f/School work/A Level/Computer Science/MML-To-Midi-Project/function_tests
Andy@INP-Carbine /f/School work/A Level/Computer Science/MML-To-Midi-Project/function_tests
$ gcc --std=c99 writeVariableLengthQuantity_test.c&&./a.exe
Input n Expected Expected Actual   Actual
      VLQ      Return   VLQ      Return
00000000 00000000 00000000 00000000 00000001 PASS
00000040 00000040 00000001 00000040 00000001 PASS
0000007F 0000007F 00000001 0000007F 00000001 PASS
00000080 00000081 00000002 00000081 00000002 PASS
00002000 000000C0 00000002 000000C0 00000002 PASS
00003FFF 00007FFF 00000002 00007FFF 00000002 PASS
00004000 00008081 00000003 00008081 00000003 PASS
00100000 000080C0 00000003 000080C0 00000003 PASS
001FFFFFF 007FFFFFF 00000003 007FFFFFF 00000003 PASS
00200000 00808081 00000004 00808081 00000004 PASS
08000000 008080C0 00000004 008080C0 00000004 PASS
0FFFFFFF 7FFFFFFF 00000004 7FFFFFFF 00000004 PASS

NOTE: All VLQ values corrected for endianness

Andy@INP-Carbine /f/School work/A Level/Computer Science/MML-To-Midi-Project/function_tests
$

```


| | |
|--------------------|--|
| Function prototype | <code>int readVariableLengthQuantity(char *inputPtr)</code> |
| Purpose | Read a big endian variable length quantity from pointer "inputPtr", and return it as a little endian integer. |
| Testing code | <pre> #include <stdio.h> struct testCase { char input[4]; int expectedOutput; }; int readVariableLengthQuantity(char *inputPtr) { char *workingPtr = inputPtr; while (*workingPtr & 0x80) { workingPtr++; } int output = 0; int outputShift = 0; do { output = (*workingPtr & 0x7F) << outputShift; outputShift += 7; } while (workingPtr-- != inputPtr); return output; } int main(int argc, char *argv[]) { struct testCase tests[] = { {0x00, 0x00, 0x00, 0x00, 0}, {0x40, 0x00, 0x00, 0x00, 0x40}, {0x7F, 0x00, 0x00, 0x00, 0x7F}, {0x81, 0x00, 0x00, 0x00, 0x80}, {0xC0, 0x00, 0x00, 0x00, 0x2000}, {0xFF, 0x7F, 0x00, 0x00, 0x3FFF}, {0x81, 0x80, 0x00, 0x00, 0x4000}, {0xC0, 0x80, 0x00, 0x00, 0x100000}, {0xFF, 0xFF, 0x7F, 0x00, 0x1FFFFFF}, {0x81, 0x80, 0x80, 0x00, 0x200000}, {0xC0, 0x80, 0x80, 0x00, 0x8000000}, {0xFF, 0xFF, 0xFF, 0x7F, 0xFFFFFFFF}; int actualOutput; printf("Input Expected Actual \n"); for (int i = 0; i < sizeof(tests) / sizeof(struct testCase); i++) { actualOutput = readVariableLengthQuantity(tests[i].input); for (char c = 0; c < 4; c++) { printf("%02X", (unsigned char) tests[i].input[c]); } printf(" %08X %08X", tests[i].expectedOutput, actualOutput); if (actualOutput != tests[i].expectedOutput) { printf(" FAIL\n"); continue; } printf(" PASS\n"); } return 0; } </pre> |

| | |
|---------|---|
| Outcome |  <pre> MINGW32:/f/School work/A Level/Computer Science/MML-To-Midi-Project/function_tests Andy@1NP-Carbid /f/School work/A Level/Computer Science/MML-To-Midi-Project/function_tests \$ gcc --std=c99 readVariableLengthQuantity_test.c&&./a.exe Input Expected Actual 00000000 00000000 00000000 PASS 40000000 00000040 00000040 PASS 7F000000 0000007F 0000007F PASS 81000000 00000080 00000080 PASS C0000000 00002000 00002000 PASS FF7F0000 00003FFF 00003FFF PASS 81800000 00004000 00004000 PASS C0800000 00100000 00100000 PASS FFFF7F00 001FFFFF 001FFFFF PASS 81808000 00200000 00200000 PASS C0808000 08000000 08000000 PASS FFFFFFFF 0FFFFFFF 0FFFFFFF PASS Andy@1NP-Carbid /f/School work/A Level/Computer Science/MML-To-Midi-Project/function_tests \$ </pre> |
|---------|---|

mmltomidi Lexer and Parser Test

In order to test the parsing of the mmltomidi program, I have added the code below to its “int main” function. This code will print out the contents of the “mmlFileStruct” structure written to by the parser.

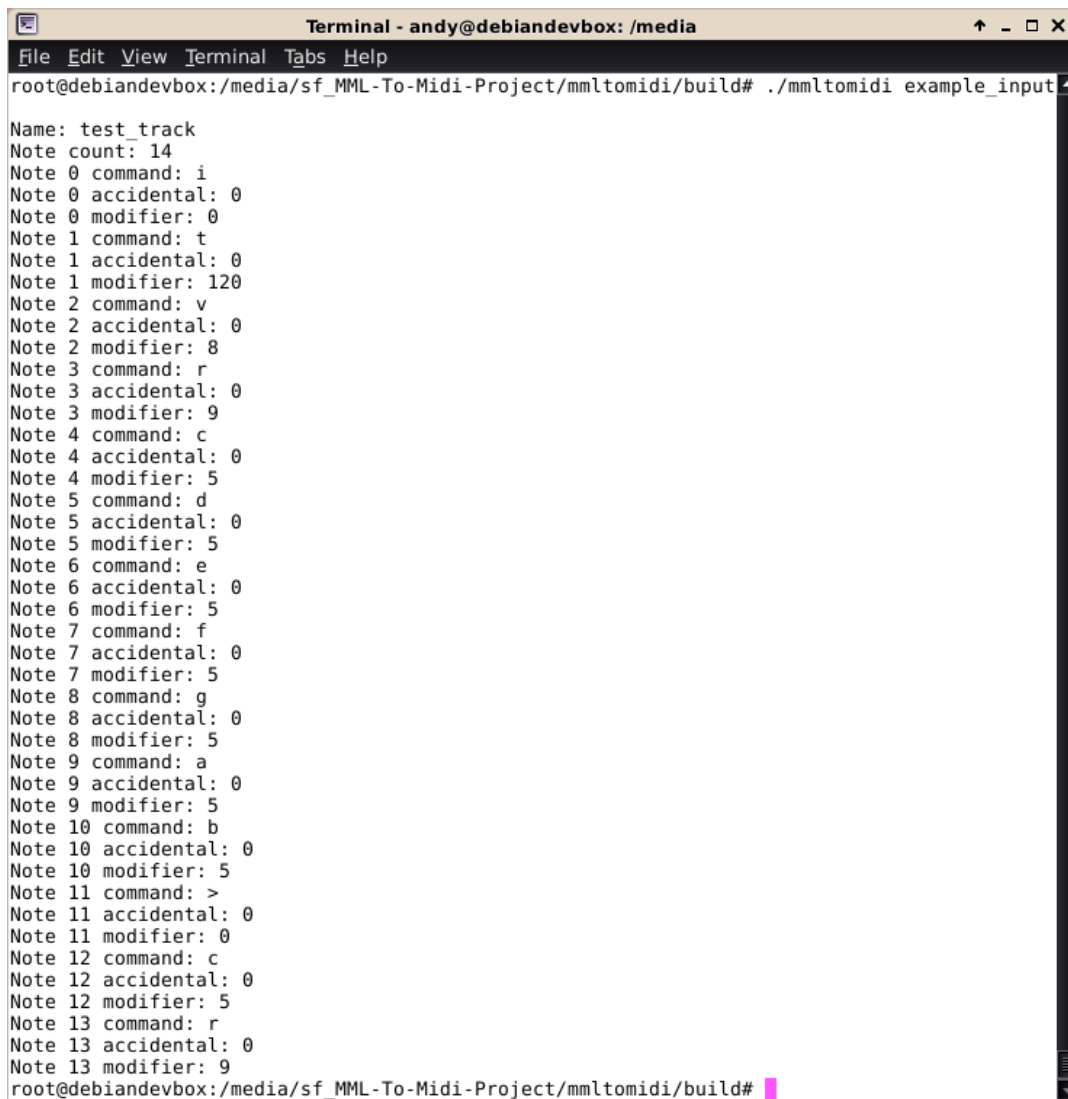
```

printf("Name: %s\n", processedMmlFile.name);
printf("Note count: %d\n", processedMmlFile.noteCount);

for (int i = 0; i < processedMmlFile.noteCount; i++) {
    printf("Note %d command: %c\n", i, processedMmlFile.notes[i].command);
    printf("Note %d accidental: %d\n", i, processedMmlFile.notes[i].accidental);
    printf("Note %d modifier: %d\n", i, processedMmlFile.notes[i].modifier);
}

```

Shown below is a screenshot of the modified mmltomidi program being run, with the example MML file from the design section as an input.

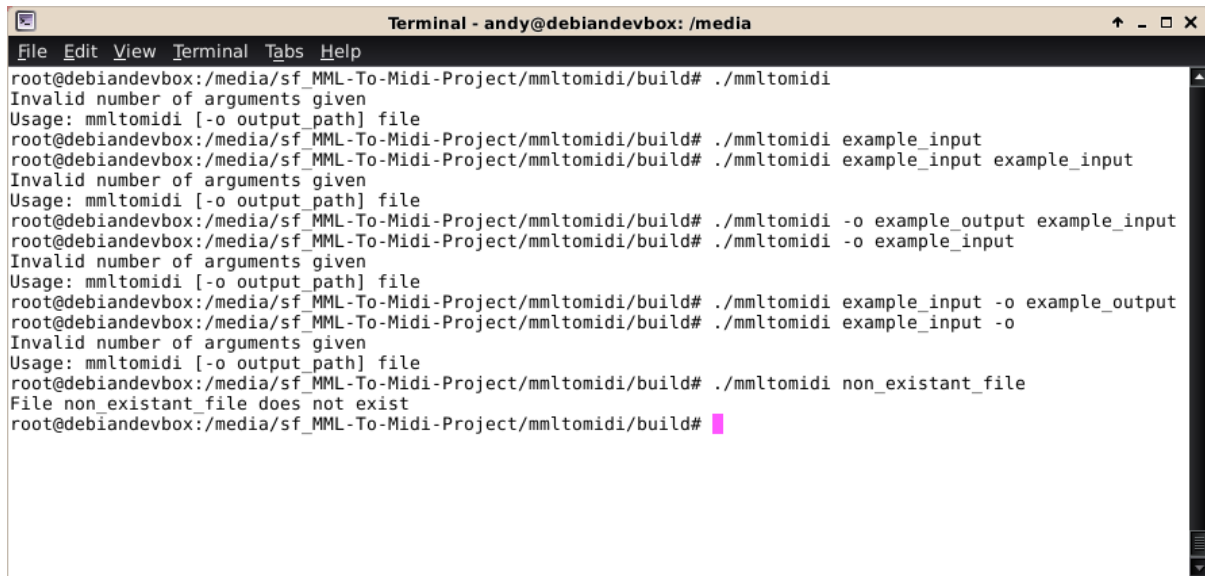


```
Terminal - andy@debiandevbox: /media
File Edit View Terminal Tabs Help
root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build# ./mmltomidi example_input
Name: test_track
Note count: 14
Note 0 command: i
Note 0 accidental: 0
Note 0 modifier: 0
Note 1 command: t
Note 1 accidental: 0
Note 1 modifier: 120
Note 2 command: v
Note 2 accidental: 0
Note 2 modifier: 8
Note 3 command: r
Note 3 accidental: 0
Note 3 modifier: 9
Note 4 command: c
Note 4 accidental: 0
Note 4 modifier: 5
Note 5 command: d
Note 5 accidental: 0
Note 5 modifier: 5
Note 6 command: e
Note 6 accidental: 0
Note 6 modifier: 5
Note 7 command: f
Note 7 accidental: 0
Note 7 modifier: 5
Note 8 command: g
Note 8 accidental: 0
Note 8 modifier: 5
Note 9 command: a
Note 9 accidental: 0
Note 9 modifier: 5
Note 10 command: b
Note 10 accidental: 0
Note 10 modifier: 5
Note 11 command: >
Note 11 accidental: 0
Note 11 modifier: 0
Note 12 command: c
Note 12 accidental: 0
Note 12 modifier: 5
Note 13 command: r
Note 13 accidental: 0
Note 13 modifier: 9
root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build#
```

Cross referencing this result with the input, it can be seen that the parser is working correctly.

mmltomidi User Interface Test

In this section, the mmltomidi program will be called in a variety of ways in order to test its user interface, and also the handling of invalid inputs. These test calls are shown in the screenshot below.



```

Terminal - andy@debiandevbox: /media
File Edit View Terminal Tabs Help
root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build# ./mmltomidi
Invalid number of arguments given
Usage: mmltomidi [-o output_path] file
root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build# ./mmltomidi example_input
root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build# ./mmltomidi example_input example_input
Invalid number of arguments given
Usage: mmltomidi [-o output_path] file
root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build# ./mmltomidi -o example_output example_input
root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build# ./mmltomidi -o example_input
Invalid number of arguments given
Usage: mmltomidi [-o output_path] file
root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build# ./mmltomidi example_input -o example_output
root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build# ./mmltomidi example_input -o
Invalid number of arguments given
Usage: mmltomidi [-o output_path] file
root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build# ./mmltomidi non_existant_file
File non existant file does not exist
root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build#

```

Looking at the tests above, the mmltomidi program appears to comply with the user interface specified in the design section, and give appropriate errors when the calling syntax is incorrect.

mmltomidi Full Integration Test

In this test, the mmltomidi program will be tested as a whole. The screenshot below shows the program being run with the example MML file shown in the design section.



```

Terminal - andy@debiandevbox: /media
File Edit View Terminal Tabs Help
root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build# ls
example_input lex.yy.c mmltomidi mmltomidi.dSYM y.tab.c y.tab.h
root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build# ./mmltomidi example_input

root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build# ls
example_input lex.yy.c mmltomidi mmltomidi.dSYM output.midi y.tab.c y.tab.h
root@debiandevbox:/media/sf_MML-To-Midi-Project/mmltomidi/build#

```

The hex dump of the MIDI file outputted from this execution is shown below.

```

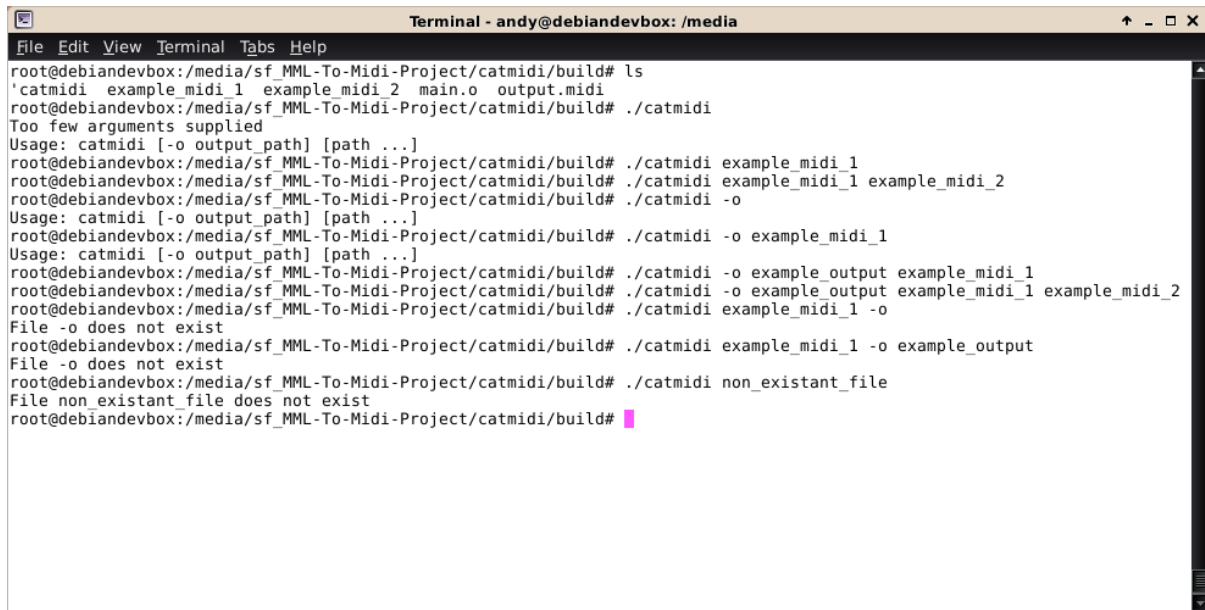
4D 54 68 64 00 00 00 06 00 00 00 01 00 08 4D 54 72 6B 00 00 00 7E 00 FF 03 0A 74 65 73 74
5F 74 72 61 63 6B 00 FF 58 04 04 02 18 08 00 FF 51 03 03 D0 90 00 C0 00 00 C0 00 00 FF 51
03 03 D0 90 00 90 00 00 20 80 00 00 00 90 3C 70 08 80 3C 70 00 90 3E 70 08 80 3E 70 00 90
40 70 08 80 40 70 00 90 41 70 08 80 41 70 00 90 43 70 08 80 43 70 00 90 45 70 08 80 45 70
00 90 47 70 08 80 47 70 00 90 48 70 08 80 48 70 00 90 00 00 20 80 00 00 00 FF 2F 00

```

Cross referencing against the expected MIDI file to be outputted by the program in the design section, the program can be verified to be giving the correct output.

catmidi User Interface Test

In this section, the catmidi program will be called in a variety of ways in order to test its user interface, and its behaviour towards invalid calling syntax. These tests are shown in the screenshot below.

A terminal window titled "Terminal - andy@debiandevbox: /media" showing a series of commands and their outputs for the catmidi program. The commands and outputs are as follows:

```
root@debiandevbox:/media/sf_MML-To-Midi-Project/catmidi/build# ls
'catmidi example_midi_1 example_midi_2 main.o output.midi
root@debiandevbox:/media/sf_MML-To-Midi-Project/catmidi/build# ./catmidi
Too few arguments supplied
Usage: catmidi [-o output_path] [path ...]
root@debiandevbox:/media/sf_MML-To-Midi-Project/catmidi/build# ./catmidi example_midi_1
root@debiandevbox:/media/sf_MML-To-Midi-Project/catmidi/build# ./catmidi example_midi_1 example_midi_2
root@debiandevbox:/media/sf_MML-To-Midi-Project/catmidi/build# ./catmidi -o
Usage: catmidi [-o output_path] [path ...]
root@debiandevbox:/media/sf_MML-To-Midi-Project/catmidi/build# ./catmidi -o example_midi_1
Usage: catmidi [-o output_path] [path ...]
root@debiandevbox:/media/sf_MML-To-Midi-Project/catmidi/build# ./catmidi -o example_output example_midi_1
root@debiandevbox:/media/sf_MML-To-Midi-Project/catmidi/build# ./catmidi -o example_output example_midi_1 example_midi_2
root@debiandevbox:/media/sf_MML-To-Midi-Project/catmidi/build# ./catmidi example_midi_1 -o
File -o does not exist
root@debiandevbox:/media/sf_MML-To-Midi-Project/catmidi/build# ./catmidi example_midi_1 -o example_output
File -o does not exist
root@debiandevbox:/media/sf_MML-To-Midi-Project/catmidi/build# ./catmidi non_existant_file
File non existant file does not exist
root@debiandevbox:/media/sf_MML-To-Midi-Project/catmidi/build#
```

Observing the tests above, it can be seen that the catmidi programs user interface functions as described in the design section of this document.

catmidi Full Integration Test

The catmidi program will be tested as a whole in this section. The screenshot below shows a call to the program with inputs "example_midi_1.midi" and "example_midi_2.midi".

A terminal window titled "Terminal - andy@debiandevbox: /media" showing a single command and its output for the catmidi program. The command and output are as follows:

```
root@debiandevbox:/media/sf_MML-To-Midi-Project/catmidi/build# ./catmidi example_midi_1.midi example_midi_2.midi
root@debiandevbox:/media/sf_MML-To-Midi-Project/catmidi/build#
```

The MML code used to generate these example inputs and their hex dumps are shown in the tables below.

| example_midi_1.midi | |
|---------------------|---|
| MML code: | <pre>##Example comment #name example #instrument 0 #tempo 120 \$c cdefgab>c play r9\$cr9</pre> |
| Hex dump: | <pre>4D 54 68 64 00 00 00 06 00 00 00 01 00 08 4D 54 72 6B 00 00 00 7B 00 FF 03 07 65 78 61 6D 70 6C 65 00 FF 58 04 04 02 18 08 00 FF 51 03 03 D0 90 00 C0 00 00 C0 00 00 FF 51 03 03 D0 90 00 90 00 00 20 80 00 00 00 90 3C 7F 08 80 3C 7F 00 90 3E 7F 08 80 3E 7F 00 90 40 7F 08 80 40 7F 00 90 41 7F 08 80 41 7F 00 90 43 7F 08 80 43 7F 00 90 45 7F 08 80 45 7F 00 90 47 7F 08 80 47 7F 00 90 48 7F 08 80 48 7F 00 90 00 00 20 80 00 00 00 FF 2F 00</pre> |

| example_midi_2.midi | |
|---------------------|---|
| MML code: | <pre>##Example comment #name example #instrument 0 #tempo 120 \$c cdefgab>c play r9>\$cr9</pre> |
| Hex dump: | <pre>4D 54 68 64 00 00 00 06 00 00 00 01 00 08 4D 54 72 6B 00 00 00 7B 00 FF 03 07 65 78 61 6D 70 6C 65 00 FF 58 04 04 02 18 08 00 FF 51 03 03 D0 90 00 C0 00 00 C0 00 00 FF 51 03 03 D0 90 00 90 00 00 20 80 00 00 00 90 48 7F 08 80 48 7F 00 90 4A 7F 08 80 4A 7F 00 90 4C 7F 08 80 4C 7F 00 90 4D 7F 08 80 4D 7F 00 90 4F 7F 08 80 4F 7F 00 90 51 7F 08 80 51 7F 00 90 53 7F 08 80 53 7F 00 90 54 7F 08 80 54 7F 00 90 00 00 20 80 00 00 00 FF 2F 00</pre> |

Finally, a hex dump of the combined MIDI outputted by the catmidi program is shown below.

```
4D 54 68 64 00 00 00 06 00 00 00 01 00 08 4D 54 72 6B 00 00 00 DB 00 FF 03 07 65 78 61 6D
70 6C 65 00 FF 58 04 04 02 18 08 00 FF 51 03 03 D0 90 00 C0 00 00 C0 00 00 FF 51 03 03 D0
90 00 90 00 00 00 FF 51 03 03 D0 90 00 C1 00 00 C1 00 00 FF 51 03 03 D0 90 00 91 00 00 20
80 00 00 00 90 3C 7F 00 81 00 00 00 91 48 7F 08 80 3C 7F 00 90 3E 7F 00 81 48 7F 00 91 4A
7F 08 80 3E 7F 00 90 40 7F 00 81 4A 7F 00 91 4C 7F 08 80 40 7F 00 90 41 7F 00 81 4C 7F 00
91 4D 7F 08 80 41 7F 00 90 43 7F 00 81 4D 7F 00 91 4F 7F 08 80 43 7F 00 90 45 7F 00 81 4F
7F 00 91 51 7F 08 80 45 7F 00 90 47 7F 00 81 51 7F 00 91 53 7F 08 80 47 7F 00 90 48 7F 00
81 53 7F 00 91 54 7F 08 80 48 7F 00 90 00 00 00 81 54 7F 00 91 00 00 20 80 00 00 00 FF 2F
00
```

Examining this combined file, it can be seen that the catmidi program functions as it should. Also, listening to the output with a MIDI player, I can hear that the two input MIDI files have been correctly combined.

Evaluation

Completion of the Objectives

| Objective | Evaluation |
|--|--|
| A program should be written that takes text file containing a variant of the MML as an input, and outputs a single-channel MIDI file that can be play with conventional software. | I completed this objective through the writing of the mmltomidi program. |
| A program should be written that takes multiple MIDI files generated by the aforementioned program and combines them into a single multi-channel MIDI file. This combined MIDI file should be playable using conventional software also. | I completed this objective with the writing of the catmidi program. |
| The programs written for this project should: <ul style="list-style-type: none"> • Complete their execution in under one second, so as to not interrupt the users work flow • Use less than half a megabyte of memory during execution • Be a less than a quarter of a megabyte in size | I met all of these objectives: <ul style="list-style-type: none"> • Both programs complete their execution in well under one second, even when compiled without any optimisation. • While it depends on the size of the files given as inputs to the programs, both programs generally use well under 512 kilobytes of memory. • Combined, the programs take up less than 60 kilobytes of disk spaces |

| | |
|---|---|
| <p>A version of the music macro language should be designed that will be used as the input for the program that generates a single-channel MIDI file from a single MML file. This language should:</p> <ul style="list-style-type: none"> • Have all the functionality of existing variants of the MML, including support for: <ul style="list-style-type: none"> ○ Octave changing ○ Accidentals ○ Default length setting ○ Volume changing ○ Tempo setting ○ Macros • Have as unambiguous a syntax as possible, with a clear logical progression | <p>I completed this objective in my design section. The language I designed there exceeds the functionality of most MML versions with the inclusion of transposition. It also is necessarily unambiguous as a BNF grammar had to be written for it.</p> |
|---|---|

Evaluation of Development

I believe that the development of the project went well – the use of Lex, Yacc and makefiles all proved to be good decisions, as each saved me a great deal of time. An area that was notably lacking during development however was testing. I feel that I could have done more during the writing of my code.

Also, it would have been very useful to have a proper debugging facility during development, as I am confident this would have saved time.

Feedback from Potential Users

I gave a copy of the programs developed for this project to a musical friend of mine for evaluation, and they did not have any complaints having used them to convert some sheet music. The macros and other features of the language apparently proved invaluable, as they reduced the redundancy in the language greatly. Overall I feel that this evaluation confirms that the user orientated objectives for the project have been completed.

What I Would Change

If I were to do the project again I would implement a better system for testing my programs, though I am still unsure of how this would be achieved. I would also consider the idea of developing a single program to solve the problem, as having two programs to use has proved to be tedious at times. Finally, I would like to add more functionality to the music description language designed for the project, and perhaps make it more similar to a programming language: this could be achieved with the inclusion of loops, variables and control structures.

Bibliography

- [1] The MIDI Manufacturers Association
The Complete MIDI 1.0 Detailed Specification
Version 96.1, Third Edition (1996)

- [2] John R. Levine, Tony Mason & Doug Brown
lex & yacc
Second Edition (1995)

- [3] Brian W. Kernighan, Dennis M. Ritchie
The C Programming Language
Second Edition (1988)