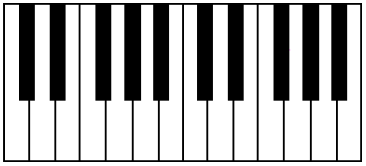
DEMUSE



MIDI MUSIC PROGRAMMING DSL

DEMUSE is an experimental toy programming language that can be used to compose and generate music in the MIDI format. It is mainly intended for **programmers** interested in music. Though it can and may also be used by musicians.

# Context

DEMUSE is mainly an experiment to find out whether music can be made more accessible to programmers when it’s used in a way that programmers are used to.

It is also trying to make simple audio/music programming more accessible by providing a ready to go environment.

# Goal

DEMUSE tries to merge the fun of programming with the fun of music using a specialized DSL, making it compact and easy to use.

Being a DSL, DEMUSE is not stuck using standard programming constructs and therefore allows to skip a lot of boilerplate code. Which in turn, allows the programmer to go straight to the music.

# FEatures

* Language integrated musical notation inspired by the ABC-notation and staccato.
* Standard chords, scales and arpeggios and patterns are provided.
* Or define your own reusable chords, scales, arpeggios and patterns.
* Full programming language inspired by C-style languages.
* Easy-to-use IDE with direct visual feedback of the compiled MIDI.
* Compiles directly to MIDI-files.
* Or play them from the DEMUSE player!
* Available for Windows, Mac OS and Linux.

# Syntax

The syntax representing any musical notations is inspired by Staccato and the ABC-notation. While the programming related constructs are inspired by the C programming language.

The following DEMUSE code can be used to play a simple C-major scale once:

track HelloWorld {  
 <C D E F G A B>;  
}

This code could be refactored to the following code which would play the exact same C-major scale once:

track HelloWorld {  
 var n = C;  
 do { <n++>; } while(n != C);  
}

# Structure

Each DEMUSE score (a DEMUSE source file) contains any number of tracks living at the root of the score creating their own scope.

Any global variables or code may be defined outside these tracks to initialize or populate global variables. Though no notes may be played here.

Inside the tracks code and musical notes can be combined to easily create bass patterns or even generate melodies.

# Implementation

## Tokenization

During tokenization language construct is transformed to a symbol. Ambiguity may arise between notes and variables (e.g. note C4 and variable ‘C4’), therefore variables names may not be named anything that may also be a valid note. In other words, when such ambiguity arises, a note is assumed.

## Parsing

The list of tokens is parsed into an abstract syntax tree (AST). Any mistokenization due to ambiguity may cause an error when e.g. a note is placed in a place where it is not allowed like an if statement.

The resulting AST could technically be compiled to any format like a tracker module, a waveform etc. Or may even be directly interpreted.

## Compiling

By default DEMUSE compiles a score to a MIDI file by transforming the AST into appropriate MIDI messages.

Since the actual compiler is implemented similarly to a virtual machine new errors may arise that could not have been detected sooner.

INTERPRETING

A score may also be directly interpreted by the virtual machine acting as a DEMUSE player. By directly sending the MIDI messages to a midi device instead writing them to a file.

# IDE\*

The DEMUSE language is accompanied by a very simple IDE that helps in the development of DEMUSE scores.

It provides autocomplete for variables and patterns, displays a visual live representation of the compiled MIDI file