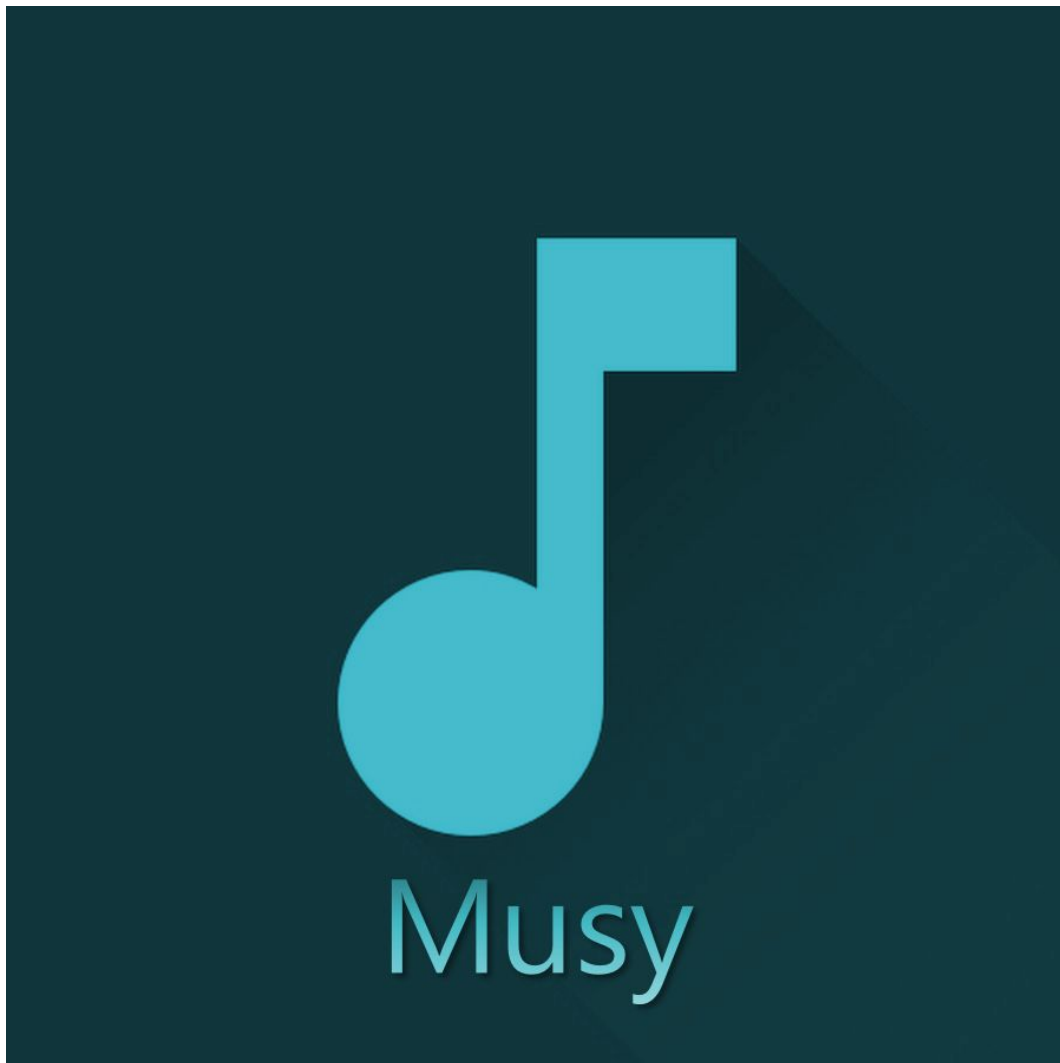


Programming Languages: Final Report



ICOM 4036-036

Prof. Wilson Rivera

Members:

Josué Albarrán - josue.albarran@upr.edu

Nelson Alemar - nelson.alemar@upr.edu

Edwin M. Rivera - edwin.rivera13@upr.edu

Final Report

Musy

Introduction

Music is something that can be created with almost anything and with a good ear. These things vary from actual musical instruments to something as simple as a table top. However, not everybody has the know-how of using a musical instrument or creating one. Some of these people may have studied music and may know how to create music but wielding an instrument is not the same thing. You can have the knowledge but if you don't have the talent or abilities, you may never be able to create music on your own. That is what we want to tackle with Musy.

On a programming standpoint, it may become complicated to generate music. Additionally, music composers may know little to nothing of how to program. With Musy, we have taken the initiative to create a programming language that simplifies this task. This will permit an user to focus more on their creativity and not in their code correctness. Even though it will be easier to code, it is recommended for the user to be familiarized with musical terminology and nomenclature. The reason for this is that if the user does not have any previous music knowledge, he will have problems identifying the meaning of certain pre-defined values that represent a note in a specific tone or the duration of it. Yet, any user is welcome to use this language.

From another perspective, this programming language could be used as a tool to learn music. The language syntax is simple enough and the learning curve isn't steep. The user could experiment with different sounds at different tones, listen to what they sound like and how to differentiate each one. It could also be used as a complementary tool for a music course. This means that users that have no music experience could take advantage of Musy, even if the primary purpose of this language is to help in composing music.

Language Tutorial


This is a step by step guide on how to use the programming language:

- I. Download Musy from the following link <https://j-albarran.github.io/musy/>
- II. Make sure Java is installed in your computer and the environment variables are set on your computer to be able to compile and run the code.
- III. Proceed to compile and run the program with a text file as input or write the code in the command line.
- IV. Create notes, phrases, and ultimately a song with the details found in the language reference manual.
- V. Play the created song or export as a MIDI file.

Language Reference Manual

This reference manual outlines the technical details found in Musy programming language.

1. Lexical Analyzer
 - Ignored tokens
 - i. “ ” - Whitespace
 - ii. “\r” - Carriage return
 - iii. “\t” - Horizontal tab character
 - iv. “\n” - New line
 - Identifiers and keywords
 - i. “Part” - Part keyword binding
 - ii. “Note” - Note keyword binding

- 
- iii. “Compose” - Compose keyword binding
 - iv. “Display” - Display keyword binding
 - v. “Play” - Play keyword binding
 - vi. “To” - To keyword binding
 - vii. “Add Note” - Adds a note to a phrase
 - viii. “New Song” - Creates a song based on the inputs given
 - ix. “New Phrase” - Creates a new phrase based on the notes
 - x. “Tempo” - Tempo keyword binding
 - xi. “1/4” - Used to set the duration of a note to quarter
 - xii. “1/2” - Used to set the duration of a note to half
 - xiii. “1/1” - Used to set the duration of a note to whole
 - Delimiters
 - i. “,”
 - ii. “(“
 - iii. “)”
 - iv. “{“
 - v. “}”
 - vi. “|” - Used to separate notes
 - vii. “:.” - Used to separate keywords and variables (variable assignment)
 - Number
 - i. `(["0"-"9"])+` - Any number from 0 to 9
 - Variables

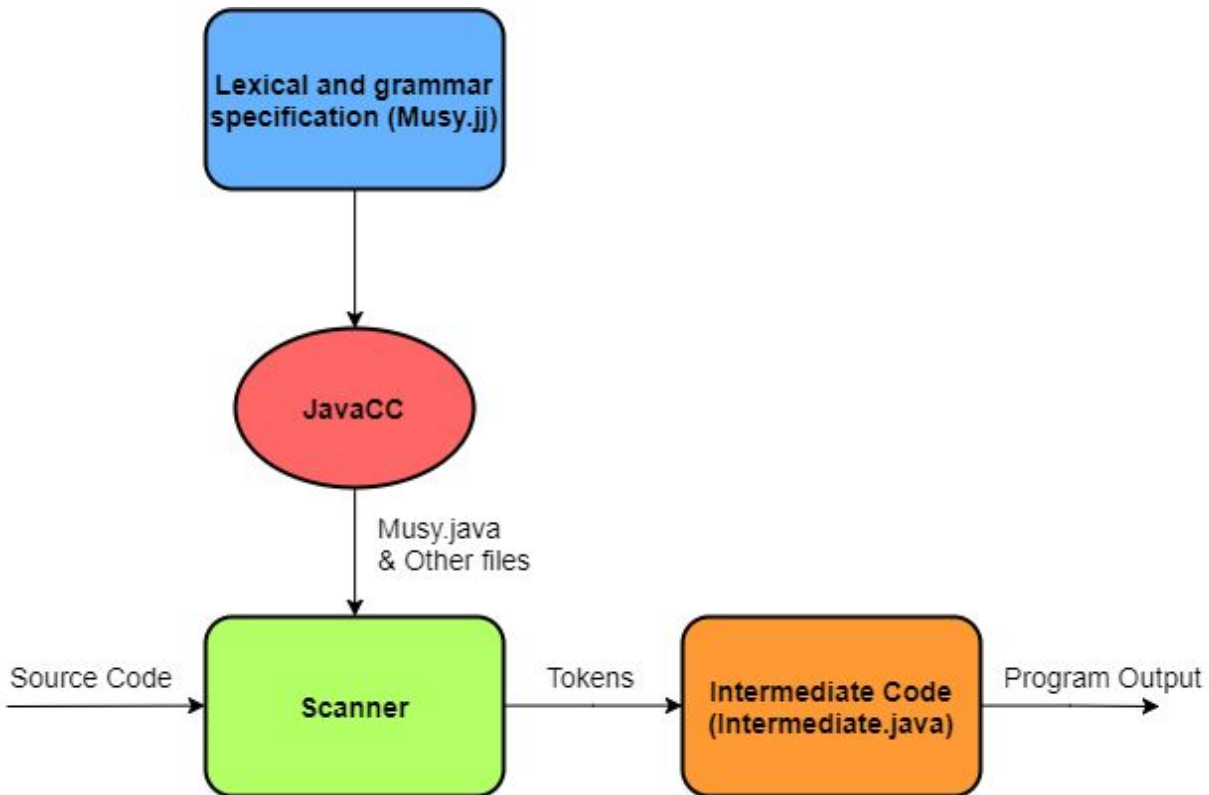
- i. `(["A"-"G"])(["0"-"8"])` - Used to set the tone
- ii. `(["a"-"z", "A"-"Z"])(["a"-"z", "A"-"Z", "0"-"9"])*` - Possible entry for variables

2. Expressions

- Define a variable
 - i. Part Lala : A4 1/4 | G2 1/2 | B0 1/1
- Compose a song
 - i. Compose : myFirstSong
- Play a song
 - i. Play : myFirstSong
- Create a phrase
 - i. New Phrase : part1
- Add notes to a phrase
 - i. Add Note A4 1/2 To part1
- Create a song
 - i. New Song : myFirstSong
- Set the tempo
 - i. Tempo : 300

Language Development


Translator Architecture



Interfaces

In the Musy translator architecture, the intermediate code is what does the heavy work of connecting the JMusic API to the tokens generated by the scanner. The scanner is in turn generated by JavaCC, which takes the *Musy.jj* file and automatically creates the scanner and parser. The *Musy.jj* file is the one that defines the grammar specifications and does the work of the lexical analyzer. Here we defined which are the accepted keywords, identifiers, delimiters, ignored characters, and expressions. This way we can ensure that whatever gets to the input of the intermediate code is in fact a correct syntax for the Musy programming language.

In the *Musy.jj* file we declared an Intermediate object which takes the tokens from the scanner and uses them to connect the JMusic API. These tokens are used to call



functions within the API such as creating songs, playing a song, adding notes, etc. Finally, the program output consists of the necessary behavior of the programming language given a certain source code.

Software Development Tools

- Java - General purpose object oriented programming language intended to develop portable code.
- JavaCC - Popular parser generator for use with Java applications. Used to convert a grammar specification to a Java program that can recognize matches to the grammar.
- JMusic library - Open source music programming language library written in Java and it is designed to assist composers and music software developers by providing support for music data structures, modifications, and I/O to various file formats. It was mainly used to develop the intermediate code.
- Git - Version Control System (VCS) for tracking changes in computer files and coordinating work on those files among multiple people. Used in this project for source code management among the team members.
- Github - Version Control Repository or web-based git. It was used to host all of the source code managed from git to the internet.


Test Methodology

During development, we tested the lexical analyzer component first with printouts to see if it was detecting the correct input as well as throwing errors when it should. This was done with redirects through the command line. Then we moved to sending data from text files as if it were a complete program. Once we got this working, the intermediate code was implemented and tested with the same input files. Further testing was performed with different cases until we got a fully functional programming language. We observed the behavior of the programming language under different scenarios.

Test Programs

Test Program 1:

Part Pow = Jo Note hey = mama



Part Coco = Jo + Ja
Note ho = yo + yi

Test Program 2:

Part Lala : A4 1/4 | G2 1/2 | B0 1/1

Test Program 3:

Add Note A3 1/2 To Momo

Test Program 4:


New Song : Jaja

Test Program 5:

New Song : myFirstSong
New Phrase : part1
Add Note A4 1/2 To part1
Add Note B6 1/1 To part1
Tempo : 120
Play : myFirstSong
Compose : myFirstSong

Test Program 6:

New Song : noMelImporta
New Phrase : part1
Add Note A4 1/2 To part1
Add Note B6 1/1 To part1
Add Note C3 1/4 To part1
New Phrase : part2
Add Note E7 1/4 To part2
Add Note F2 1/4 To part2



Add Note A5 1/4 To part2
Tempo : 300
Play : noMelImporta
Compose : noMelImporta

Conclusions

While creating Musy, we were able to get a better understanding on how a programming language and its underlying components work. We got hands-on experience on building a programming language with the help of JavaCC for the scanner and parser and the JMusic library for the intermediate code. We applied some concepts discussed in class such as lexical and syntax analyzers when developing Musy. Although the JMusic API had some good documentation, we encountered some challenges when using JavaCC as it was not as straightforward. It lacks some documentation and examples against its counterpart for python PLY. However, we were able to overcome this barrier and developed a functional programming language. Some of the ideas we had at the beginning of the project changed slightly but the concept was still the same throughout the development. We were glad to have achieved the final result of a working programming language and gained the knowledge that came from developing the language.