Introduction

I chose this project to reinforce my knowledge of Antlr grammar and the use of the tree walker pattern that we used throughout the class. My intention was to develop a simple type system while creating an interface to the JMusic library. The type system was necessarily simple: a few types implemented according to lecture slides and type checking that goes along with it.

The goal was to generate a simple audio file from a score programmatically create with my language.

Implementation

For the type system, I chose to implement a few types used commonly in JMusic - Score, Part, and Phrase - along with common ones like String and Double. (Double was necessary due to JMusic frequent use of them for method parameters. For instance, the rhythm parameter for the Note constructor requires a double.)

The interpreter follows the same pattern used throughout the class: a basic interpreter main method calling the lexer, parser, and the visitor class dispatch method. The usual tree for statements and expressions, stack for keeping track of scopes, and hash maps for tracking symbols were used. My only innovation was the use of a stack to get values from one branch of the AST to another. A leaf expression, such as NumExpr, adds a Value object containing its value to the stack. The node concerned with the value just pops the stack.

Challenges

The largest technical challenge was expressing what I wanted in the grammar. For instance, I tried adding a lexical element for the types I implemented. (It is still in the grammar file, but commented out.) Antlr did not seem to like the way I wrote the regular expressions.

I also tried to develop a simple type system through the grammar, thinking to keep it simple, rather than implement the one from the lecture slides. In the end, I had to implement what was in the slides.

If I could start over, I would spend less time trying to “imagine” a language that has a strong type system and higher order functions. The better path would be to take an example directly from the JMusic tutorials and use it to develop the interface. For the type system, I would go right to the slides and develop it sooner.

Examples

Three examples are provided.

The first shows basic language features we used all semester: assignment, a basic function, etc.

The second shows my attempt at generating a small musical phrase. Unfortunately, it did not generate. As you can see from the screenshot, it had an error.

The third shows an integer being assigned to a string variable. It worked originally, but broke sometime during my debugging the second program.

Conclusions

Conceiving and developing a language requires a lot of time. Even when you master the grammar, tree walking, and basic patterns there is a tremendous amount of debugging. Many data structures come into play: stack and hashmap for maintaining symbols tables in different scopes.

However, it gives you a sense of power that you can create your own tool for developing software. I intend to continue developing this project with the intention of taking lessons-learned to try developing an full object-oriented language.