Language Development

Structure

1. Lexical Analysis(Lazy)
2. Parser
3. Code Generation/Execution

Lexical Analysis

All methods seem to use a DFA for lexical analysis, I guess that this is for easily matching regex based tokens??? Though it does not seem to be entirely necessary. I think that I can just do some basic string checking perhaps. Think about this more.

Parser

- `LL(1)` This is very simplistic. It would be manageable, but not preferred, as it requires weird constraints on the grammar, making things a bit more complicated.This seems to be the Core for most good compilers `gcc`, `clang`, etc. Lets work with this for now, I think that it would be sufficient for the necessary purposes
- `LL(k)`/`LL(*)` There are significantly better, but also more complicated. They allow for more advanced grammars, but still have some of the pitfalls of the `LL` based parsers.
- `LR(1)` This is significantly better, and has fewer pitfalls of the `LL` parsers, but it is significantly more complicated to implement, and harder to debug, and produce debug information for the user. See if this parser can be implemented.
- `LALR(1)` This seems to be the standard for parse generators. It is used by both *Bison* and *Yacc*. Thus it must be pretty good. Older languages tend to utilize this method.
- `PEG(Packrat)` This seems to be a common implementation for parsing a specific grammar. More research needs to be done in order to learn more of its merits.