Project Journal: Weeks 8-10 (10/22 - 12/16)

BLARG! Systems - Ryan Kispert

Weekly Goals:

Week 8:

- Finalize structure of Lexer.
- Start implementing all tokens in Lexer.

Week 9:

- Finish a simple operator syntax.
- Prepare Mid-Year presentation.

Week 10:

- Finalize Mid-Year presentation.
- Present Mid-Year presentation.
- Prepare advisor meeting.

Research:

Weeks 8-10 have included some of the least research overall, as it has mostly been repetition. Since I had already managed to sort out the lexer's[1] basic process, it's primarily time-consuming to sort out how to parse each token from my file. I've primarily referenced a compiler written in C called MiniC[1], posted on GitHub by NikRadi. I've used many of the same methods to identify tokens. However, I simplified a few parts and decided to add some limited error detection at this stage of the compiler process. Presently, my `Source.c` file verifies the BLARG! file is a `.blarg` file, reads it, sets up the lexer struct[2][2], and then runs the `LexSource(Lexer* l)` function. The resulting Token array is printed to the console for debugging purposes to ensure the file was tokenized properly. The `Lexer.c` file contains the actual lexer, which initializes a Token struct for each token in the source file. The tokens are any meaningful part of the source file, meaning keywords, symbols, literals (such as strings, integers, and booleans), and identifiers (the names used for variables, functions, etc). Unneeded parts of the source file are ignored, such as whitespace (spaces and tabs). The lexer runs through the source file, finding the necessary parts and creating Token structs that use an enum[3][2] to say what kind of token it is. If the token is a literal or identifier, the Token is given the

[1] Part of a compiler which "tokenizes" the source code, making it easier to use later in the process
[2] A grouping of variables in C
[3] A variable type that allows programmers to use names to represent numbers.

value of that literal or the name of that identifier. These tokens are all put in an array, which will be passed to the parser, which will create an Abstract Syntax Tree (AST) that allows for easier representation and building of the final code.

Accomplishments:

- Lexer is now through multiple iterations of design
- The intended structure of Lexer is decided and mostly complete
- Mid-Year project presentation created and complete

Reflection:

Having realized this project's complexity, I have understandably readjusted my timeline again. I will be writing the code generation part of the BLARG! compiler in March, targeting the C language. The lexer is progressing smoothly and I expect to finish soon. Building an AST will be the next milestone of this project, allowing an easier transition to building code from BLARG!. I will still have a partially functioning prototype by the presentation date at this pace, and I am not particularly concerned about time at this point.

Bibliography:

1. *NikRadi. (n.d.). Nikradi/minic: A C-compiler written in C. GitHub. https://github.com/NikRadi/minic/tree/master*
2. *C standard library reference tutorial. Online Tutorials, Courses, and eBooks Library. (n.d.). https://www.tutorialspoint.com/c_standard_library/index.htm*