Project Journal: Weeks 5-7 (10/22 - 12/16)

BLARG! Systems - Ryan Kispert

Weekly Goals:

Week 5:

- Re-evaluate timeline to adjust for project planning changes.
- Create syntax plan.
- Research lexing algorithms.

Week 6:

- Finalize syntax plan.
- Research open-source lexers to base my version on.
- Begin working on lexer.

Week 7:

- Research references for lexing and tokenizing programs.
- Continue to develop lexer.
- Modulize lexer for easier development.

Research:

I have been deeply focused on the structures behind programming. Much of BLARG!'s structure has been based on Javascript, as I'm fairly familiar with the language. BLARG! is structured to be similar to other languages, although with a few quirks just to be unique, my personal favorite feature being all lines ending in a smiley-face :). The syntax plan was rather quick, primarily just referencing what would be critical for a simple language. To proceed with my lexer, I elected to write it in C, a language I have little experience in, as it is significantly faster and operates closer to assembly, which should additionally help me understand NASM better. As such, I consistently reviewed a C tutorial[4] and made some testing programs. A significant difference in C compared to other, higher-level languages, is manual memory management. Other languages will automatically assign and clean up memory for all variables as needed, however, the developer must do much of this themselves in C. Due to my inexperience in this (even though it is critical for Assembly), I struggled for quite some time to include reasonable memory management for reading a file, going through additional documentation for C to do so[2]. Of course, there *must* be plenty of simple issues in programming such as just printing a string, which I resolved after some frustration[5]. I researched custom data structures in C, referred to as enums[3]. I'll use these to determine the tokens my lexer

produces, which are identifiers for each part of the code that are then passed to the parser. The parser will read over these tokens and construct the the final program in the target language, in my case, Assembly x64.

Accomplishments:

- Timeline adjusted for shifts in project plan
- Syntax plan created with example programs
- Found an open-source C C compiler to reference for my compiler (MiniC)
- Learned the structure of the C language
- Began developing lexical analyzer
- Established enum references for the tokenizer part of the lexical analyzer

Reflection:

I've completed a solid amount of progress on my project these weeks. My lexer has a solid start, and I intend to finish it over the holidays. Assuming I continue on this pace, I expect to be able to generate C code from my language by mid February, allowing me plenty of time to integrate Assembly generation.

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*
3. *GeeksforGeeks. (2022, May 24). Enumeration (or enum) in C. GeeksforGeeks. https://www.geeksforgeeks.org/enumeration-enum-c/*
4. *C tutorial. Online Tutorials, Courses, and eBooks Library. (n.d.-b). https://www.tutorialspoint.com/cprogramming/index.htm*
5. *Stack Exchange (1967, March 1). Caesar : Extra random characters at the end of a char ciphertext array. CS50 Stack Exchange. https://cs50.stackexchange.com/questions/40701/caesar-extra-random-characters-at-the-end-of-a-char-ciphertext-array*