Background

Compilers and interpreters underly almost every programming language on the market. The theoretical framework (formal language theory) that back these compilers are often misunderstood by developers. Compilers, on the surface, just seem like a black box that you feed code and it produces arbitrary code in another language.

With this project I set out to build my own language compiler from scratch. Meaning that I didn't want to use libraries like LLVM or parser generators such as YAAC, ANTLR, or Bison.

I found this area of computer science interesting and wanted to understand how these systems work fundamentally. Through the creation of midnight compiler I've learned so much about context-free languages, parsing algorithms, code generators, and lots of other abstract concepts.

Technologies

Accomplishments

- Generates code for large subset of C
- Lexer
  - Flexible Tokenization of C code
  - Handles (almost) all c data types
- Parser
  - Successfully builds IR (AST) for C code
  - Handles for loops, while loops, structures, unions, pointers
- Generator
  - Generates x86 Assembly for C code.
  - Code runs on any computer running on x86/x64 Architecture

Challenges

- Little experience in this area of CS
- C language's nuances and constructs (stack alignment, pointer depth, structs, etc.)
- Creation of an efficient parser for the compiler
- Handling of stack frames, function calls, and scopes
- Error handling and debugging information

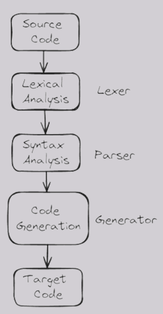
Future Work

- Expand on Subset of C supported
- Allow generation for other architectures (x64 ASM, ARM, etc.)

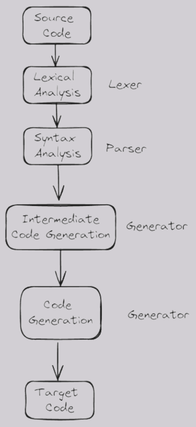
System Overview

Compilers are the result of multiple interconnected subsystems. These components utilize the output produced by preceding components.

These components are highlighted in the images below.



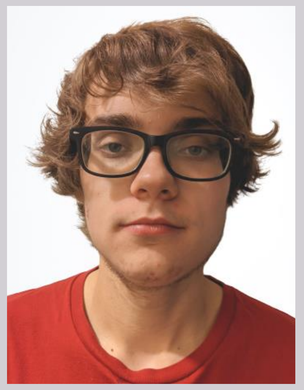
Or



MIDNIGHT COMPILER

Github Repo

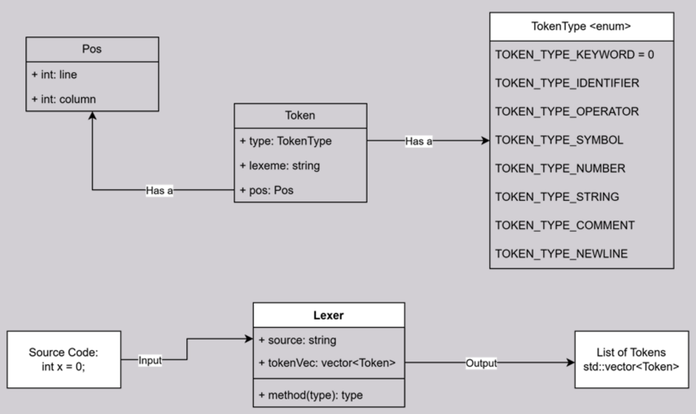
Group (solo)



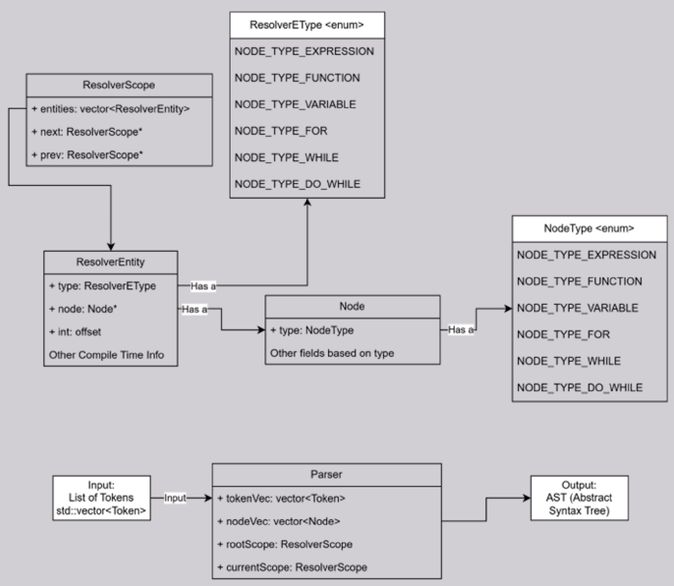
troxeldj@mail.uc.edu / CS

Stage: Lexical Analysis
Component: "Lexer"
Input: Raw text/code
Output: Linear "list" of Token structures

Lexical analysis is concerned with creating a linear list of Tokens from input text (usually in the form of a file with the languages extension). This process is often referred to as "Tokenization".



Stage: Parsing
Component: "Parser"
input: List of Tokens (output from Lexer)
Output: AST (Abstract Syntax Tree) - a structure hierarchically representing the source code.



Stage: Code Generation
Component: "Generator" or "Translater"
input: AST (output from Parser)
Output: Code in another format (x86 Assembly in this example