Michael Bozhilov

Senior Project I

Fall 23/24

Last page: 51

# Nulascript

**Nulascript** is a dynamically typed programming language that allows memory access through pointers. Pointers represent a crucial concept in computer programming, particularly in low-level languages like C. They store memory addresses rather than actual values. This distinction becomes important in terms of efficiency because it allows you to work with data in memory directly without making unnecessary copies. For example, in high-level scripting languages running on virtual machines and in deployment pipelines, avoiding data duplication and having direct access to memory locations can lead to faster execution and more cost-effective infrastructure management.

The process of running **Nulascript** code involves several stages. At a high level, it starts with the user writing code in a text file. This code is then passed to a lexer, which breaks it down into tokens. These tokens are then used as input for a parser to construct an abstract syntax tree. Finally, the AST is interpreted to execute the code.

The lexer, also known as a lexical analyzer, is a core component of a compiler or interpreter. Its primary role is to examine the source code of a programming language and divide it into individual tokens. These tokens are the smallest meaningful units of code, like keywords, identifiers, operators, and literals. The lexer categorizes these tokens and sends them to subsequent stages in the compilation or interpretation process for further analysis, where the code's structure and meaning are further understood and processed. Essentially, a lexer serves as the initial step in the process of translating human-readable source code into a format that a computer can comprehend and work with.

The parser, also known as a syntactic analyzer, is another core component which primary task is to inspect the source code tokens generated by the lexer and arrange them into a hierarchical structure known as the syntax abstract tree. The tree servers as a representation of the source code which clearly defines the relationship between different tokens and their grammatical structure. The parser's role is to verify whether the code complies with the language's syntax rules and to generate a structured representation that can be further processed by the compiler or interpreter. Additionally, there exist various types of parsers. In the context of **Nulascript**, the tokens are processed by a Pratt parser, also referred to as a top-down operator precedence parser. Our parser primarily begins at the top of the parse tree, examining the initial symbol of the grammar, and proceeds linearly by inspecting the subsequent tokens to determine the appropriate path to take. This approach leads to efficient parsing. We'll delve into more detail about the parser's implementation as we continue.

An interpreter is, as the name implies, the underlying software which interprets the meaning of our code by walking the abstract tree we’ve just created. Interpreters provide a direct way to execute commands without building a machine code executable. Rather it executes the code on the fly which enables us to interactively run commands, if needed. At a high level, our **Nulascript** code will run from a C++ binary, which breaks down into machine code and contains instructions on how to execute each step from our tree.