Compiler

Paolo Bettelini

Contents

1 Compiler

The life of a program looks as follows:

$$\textbf{Source code} \rightarrow \textbf{compiler} \rightarrow \begin{matrix}\textbf{bytecode}\\ \textbf{machine code}\end{matrix} \rightarrow \begin{matrix}\textbf{VM}\\ \textbf{CPU}\end{matrix}$$

However the compiler is not straightforward

$$\textbf{Source code} \rightarrow \overbrace{\boxed{\textbf{Lexer} \rightarrow \textbf{Parser} \rightarrow \textbf{Validator} \rightarrow \textbf{Assembler}}}^{\text{Compiler}} \rightarrow \begin{matrix}\textbf{Bytecode}\\ \textbf{Machine code}\end{matrix} \rightarrow \begin{matrix}\textbf{VM}\\ \textbf{CPU}\end{matrix}$$

1.1 Lexer

The lexer takes the sources code and produces a list of tokens.

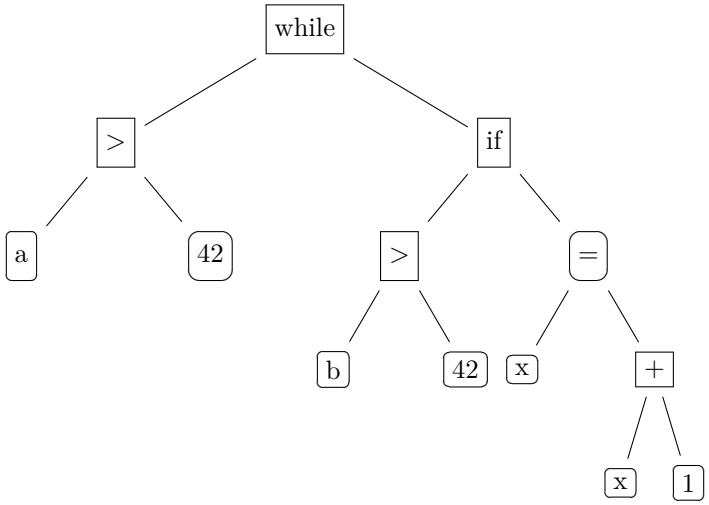
Here's an example of token design.

```
[
    {
        "type": "identifier",
        "value": "function"
    },
    {
        "type": "identifier",
        "value": "if"
    },
    {
        "type": "identifier",
        "value": "while"
    },
    {
        "type": "literal",
        "value": 42
    },
    {
        "type": "operator",
        "value": "{"
    },
    {
        "type": "operator",
        "value": "}"
    },
    {
        "type": "operator",
        "value": ";"
    }
]
```

1.2 Parser

The parser takes as input the tokens generated by the lexer. It produces an **AST** (Abstract Syntax Tree).

Here's an example of an abstract syntax tree.



1.3 Validator

The validator checks for any error in the AST. It checks if every variable exists, function calls are valid and so on.

1.4 Assembler

The assembler takes the AST and produces the final compiled code.