# Compiler Manual

David Polar
CpS 450
May 3, 2022

## Overview

This project translates Blink files to Nix assembly with the help of ANTLR. It receives a Blink file and performs lexical, syntactic, and semantic analysis, finishing with code generation. When passing through the lexer, tokens are extracted from the Blink file. This process outputs a stream of lexical tokens, which gets fed into the syntactical analysis phase. This uses a grammar to ensure that the stream of lexical tokens is arranged in a syntactically acceptable manner. The result of this process is a Parse tree. The parse tree is passed to the semantic checker, which ensures correct operations and types among the nodes of the parse tree. After the program passes all the syntactic and semantic checks, the assembly code is generated.

## Time Log

Phase 1: 11 hours

Phase 2: 10 hours

Phase 3: 8 hours

Phase 4: 12 hours

## Phase 1

Lexical analysis is the first phase of the compiler. Lexer rules are written in the G4 file to be used by ANTLR. These rules are written in two forms—fragments and tokens. Tokens are composed of either previously declared token fragments or strings.

```
fragment VALID_ID_START: ('a' .. 'z') | ('A' .. 'Z') | '_';
```

Fragment Example

```
COMMENT: '#' NOT_NEWLINE* -> skip;
```
Token Example

ANTLR passes through the given Blink file and uses the REGEX expressions to extract all the information as tokens. This is implemented in Java thus—

```
CharStream input = CharStreams.fromStream(IOUtils.toInputStream(contents,
Charset.defaultCharset()));
BlinkLexerImpl lexer = new BlinkLexerImpl(input);
CommonTokenStream tokens = new CommonTokenStream(lexer);
```
Lexer Java Implementation

Tokens found in the Blink file but not in the lexer rules are reported as errors.

## Phase 2

Syntactic analysis, or parsing, is the second phase of the compiler. These parsing rules are also written in the G4 file as REGEX expressions. However, these are written as a hierarchy, as the goal of this phase is to create a parse tree. For example—

```
start: children+=primary+ EOF;
primary: cls=blink_class | decl=declaration;
```
Parsing Rules Example

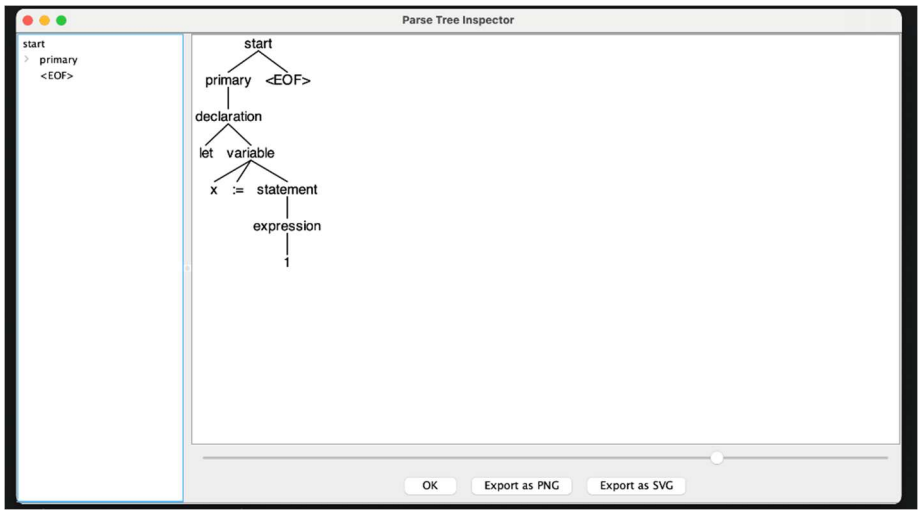
After the start node, everything is classified as a primary. Everything that is counted as a primary is either a blink_class or declaration. The parser is generated in Java using the tokens outputted from the lexer phase.

```
BlinkParser parser = new BlinkParser(tokens);
```
Parser Java Implementation

A graphical representation of the parse tree is then created with—

```
Trees.inspect(tree, parser);
```

Parse Tree Creation




Graphical Parse Tree Example

# Phase 3

Semantic analysis is the third phase of the compiler. This uses a Java implementation of the Visitor Design pattern to systematically visit every node of the parse tree. This process adds everything to the symbol table and ensures operations are correct. For example, here is the method for ensuring an Int type is correct.

```
public Object visit(TypeNode.Int node) {
    node.getArrayIndicies().forEach(i -> {
        if (i != null) {
            Type iType = (Type) visit(i);
            if (!Type.INT.equals(iType)) {
                ErrorReporter.get().reportError(node.getLineNumber(), "Int
indices must be of type int", ErrorReporter.ErrorType.SEMANTIC);
            }
        }
    });
    return Type.INT.withDimensions(node.getArrayIndicies().size());
}
```

Semantic Checking Example

# Phase 4

Code generation is the fourth phase of the compiler. This also uses the Visitor Design pattern to visit the nodes. The symbol table is also passed into this phase. It outputs Nix assembly as a .s file. For example, this is the code generated when a variable is declared—

```
public String visit(DeclarationNode.VariableDeclaration node) {
    String s = "";
    s += "# visit DeclarationNode.VariableDeclaration\n";

    s += visit(node.getValue());
    s += "\tpopq\t_" + node.getIdentifier() + "(%rip)\n";

    return s;
}
```

Variable Declaration Code Generation