Testing Plan

This document defines the testing plan for the YARTBML language. ## Introduction The YARTBML is a language designed for simplicity and ease of understanding. As with any programming language, ensuring the correctness and reliability of its components is crucial. In this document, we outline the testing strategy for the main components of the YARTBML language: Lexer, Parser, AST (Abstract Syntax Tree), and Evaluator.

Components to Be Tested

The main components of the YARTBML language that need to be tested are: - Lexer: Responsible for tokenizing the input source code. - Parser: Converts the tokenized input into an AST representation. - AST (Abstract Syntax Tree): Represents the structure of the parsed code. - Evaluator: Executes the code represented by the AST and produces results.

Testing Strategy

The testing strategy involves writing comprehensive test cases for each component to ensure that they behave as expected and handle various input scenarios appropriately. Each test case aims to verify the correctness, completeness, and robustness of the respective component.

Functions to be tested

Lexer

- `TestNextToken`: Does the tokenization function correctly tokenize the provided input string according to the expected token types and literals?

Parser

- `TestLetStatements`: Does it correctly parse let statements with various types of values (integer, boolean, identifier)?
- `TestReturnStatements`: Does it correctly parse return statements with various types of return values (integer, boolean, identifier)?
- `TestIdentifierExpression`: Does it correctly parse identifier expressions?
- `TestIntegerLiteralExpression`: Does it correctly parse integer literal expressions?

- `TestBooleanLiteralExpression`: Does it correctly parse boolean literal expressions?
- `TestParsingPrefixExpressions`: Does it correctly parse prefix expressions (e.g., !5, -15, !true, !false)?
- `TestParsingInfixExpressions`: Does it correctly parse infix expressions (e.g., 5 + 5, 5 - 5, 5 * 5, 5 / 5, 5 > 5, 5 < 5, 5 == 5, 5 != 5)?
- `TestOperatorPrecedenceParsing`: Does it correctly parse operator precedence in expressions?
- `TestIfExpression`: Does it correctly parse if expressions without else clauses?
- `TestIfElseExpression`: Does it correctly parse if expressions with else clauses?
- `TestFunctionLiteralParsing`: Does it correctly parse function literals?
- `TestFunctionParameterParsing`: Does it correctly parse function parameters?
- `TestCallExpressionParsing`: Does it correctly parse call expressions?
- `TestCallExpressionParameterParsing`: Does it correctly parse call expressions with parameters?

AST

- `TestString`: Does the AST produce the expected input source code of YARTBML?

Evaluator

- `TestEvalIntegerExpression`: Does it correctly evaluate integer expressions (e.g., 5, 10, -5, -10, 5 + 5 + 5 + 5 - 10)?
- `TestEvalBooleanExpression`: Does it correctly evaluate boolean expressions (e.g., true, false, 1 < 2, 1 > 2)?
- `TestBangOperator`: Does it correctly evaluate the bang operator (!) for boolean expressions?
- `TestIfElseExpressions`: Does it correctly evaluate if-else expressions?
- `TestReturnStatements`: Does it correctly evaluate return statements?
- `TestErrorHandling`: Does it correctly handle errors in expressions?
- `TestLetStatements`: Does it correctly evaluate let statements?
- `TestFunctionObject`: Does it correctly create function objects?
- `TestFunctionApplication`: Does it correctly apply function objects?
- `TestClosures`: Does it correctly handle closures?

Run Test Suite

Open project in IDE of your choosing. In the terminal confirm your working in the root directory of the project. In the terminal run the following commands:

1. cd internal
2. go test ./lexer
3. go test ./parser
4. go test ./ast

or test all of our packages by doing with verbose output

```
cd internal
go test ./... -v
```