PPS Project Report

By Huzaifa Shaikh (roll no E2410002)

1. General Overview

The code implements a basic interpreter/compiler front-end, resembling a minimalist scripting language or REPL (Read-Eval-Print Loop) environment. It supports:

- Tokenization of input strings.
- Syntax parsing.
- Expression evaluation.
- Environment variable management.
- Jump-based control flow for error handling (setjmp/longjmp).

2. Libraries and Typedefs

```
#include <iostream>
#include <vector>
#include <map>
#include <stdio.h>
#include <stdint.h>
#include <cstring>
#include <cassert>
#include <setjmp.h>
```

Purpose:

Standard libraries used for input/output, data structures, memory manipulation, and low-level control flow.

Custom typedefs provide convenience aliases for fixed-width integer types (u8, i32, etc.).

3. Tokenization System

```
enum TokenKind { ... };
class Token { ... };
```

Defines a set of token types (identifiers, literals, operators) and a Token class to represent them.

- Uses union to efficiently store different token values (strings, numbers, identifiers).
- Tokens carry their type (TokenKind) and position.

4. AST (Abstract Syntax Tree) Structure

Classes like Expr, Stmt, and their subclasses (e.g., BinaryExpr, CallExpr, AssignStmt) are defined.

Purpose:

AST nodes correspond to expressions, function calls, conditionals, assignments, etc.

Each class contains:

A Kind enum to define the node type.

methods for printing, evaluation, etc.

5. Parsing and Grammar Rules

The parser includes recursive descent functions:

`parse_expr(), parse_primary(), parse_stmt()`, etc.

Grammar Features:

- Arithmetic and logical expressions.
- Variable assignment.
- conditional expressions
- error handling

6. Evaluation Engine

```
double eval(Expr* expr);
void run_stmt(Stmt* stmt);
```

Environment:

A variable map stores values (`std::map<std::string, double>`).

Function definitions are also stored and resolved by name.

Execution:

- Interprets nodes directly
- Supports simple math operations, variables, etc.

7. Interface

```
void repl();
```

- Implements a Read-Eval-Print Loop.
- Continuously reads user input, tokenizes, parses, and evaluates it.
- Displays output or errors

8. Error Handling

Uses `setjmp()/longjmp()` for jumps:

Allows recovery from errors by jumping back.