**GPP Interpreter Documentation for Flex and Yacc**

Introduction

This document provides comprehensive details about the GPP Interpreter, a custom language interpreter developed to handle arithmetic operations on fractions and support user-defined functions.

System Overview

The GPP Interpreter is built using Flex and Bison, allowing it to tokenize input strings and parse them according to predefined grammar rules. It primarily focuses on arithmetic operations involving fractions and the definition and invocation of functions.

Lexical Analysis (`gpp\_interpreter.l`)

Tokenization

The lexer tokenizes input strings into identifiable tokens, including:

- Fraction Literals (`VALUEF`): Represented as `numerator/denominator`.

- Identifiers (`IDENTIFIER`): Names of functions or variables.

- Arithmetic Operators: `+`, `-`, `\*`, `/`.

- Parentheses: `(`, `)` for expression grouping.

- Keywords: `def`, `if`, and others for control structures.

Syntax Parsing (`gpp\_interpreter.y`)

Grammar and Parsing

The parser processes tokens using grammar rules defined for expressions, function definitions, and conditional statements. It handles:

- Arithmetic Expressions: Evaluating operations on fractions.

- Function Definitions: Parsing user-defined functions with their names, arguments, and bodies.

- Conditional Statements: Handling `if` statements with true and false branches.

Functions

The implementation includes several key functions:

- `addFractions`, `subtractFractions`, etc.: Perform arithmetic operations on fractions.

- `createFraction`: Converts numeric values to fractions.

- `reduce\_fraction`: Simplifies fractions to their lowest terms.

- `string\_to\_fraction`: Converts string representations to `Fraction` structs.

- `check\_arg\_body\_match`: Ensures that function body identifiers match the provided arguments.

Usage and Examples

To use the interpreter, input strings following the syntax rules should be provided. For instance:

- Defining a function: `(def sum x y (+ x y))`

- Calling a function: `(sum 1b2 3b4)`

- Basic arithmetic: `(+ 1b2 3b4)`

Error Handling

The system can handle and report various errors, such as:

- Syntax errors in expressions or function definitions.

- Undefined functions or mismatched arguments during function calls.

Limitations and Future Enhancements

Currently, the interpreter focuses on basic arithmetic and user-defined functions. Future enhancements could include:

- Support for more complex data types.

- Implementation of more built-in functions and control structures.

Conclusion

The GPP Interpreter represents a fundamental approach to understanding and building language interpreters, particularly focused on fraction arithmetic and function handling.