

## **G++ Language Interpreter Documentation**

### **Flex and Yacc Part**

**Overview:** An extensive description of the G++ Language Interpreter, a basic interpreter created with Flex and Yacc, is given in this document. The interpreter lets you define custom functions and is primarily concerned with handling mathematical operations, especially with fractions.

### **Linguistic Features:**

1. Fractions in Data Types: {numerator 'b' denominator} is used to represent it (e.g., {3b4} for 3/4).

- Identifiers: names of variables or functions.

2. Arithmetic Operations: Division ({/}), Multiplication ({\*}), Subtraction ({-}), and Addition ({+}).

Handles arithmetic operations between fractions through the use of fraction manipulation.

3. Definition and Use of Functions:

- Define Functions: The {`def`} keyword is used to define functions.
- Calling a function by name and passing it the necessary arguments is how functions are executed.

### **Expression Assessment Methodology**

1. Evaluation of Expressions from Left to Right :

- Expressions are assessed in a left-to-right direction.
- The evaluation of nested expressions begins with the innermost expression.

2. Operation Precedence:

- There are no defined rules for precedence.
- Priority is always given to expressions enclosed in parenthesis.

## **Rules of Scope**

1. Global Scope: A global scope defines all variables and functions. Local or nested scopes are not supported.

2. Function Parameters: The way function parameters are handled is that they are specific to the function.

- Inside the scope of the function, they take precedence over any global identifiers of the same name.

## **Mistake Handling and Resolution of Conflicts**

### 1. Error Management

- Syntax errors are handled by the {'yyerror'} function and are reported.
- Runtime Errors: These comprise invalid operation errors and division by zero.

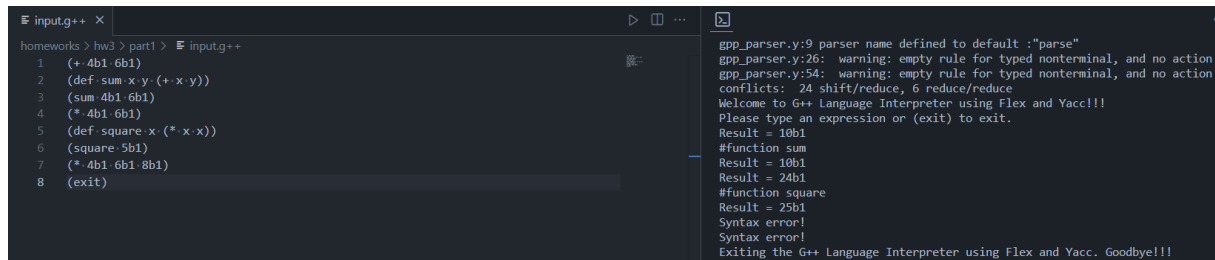
2. Resolving Conflicts: Functions take precedence over variables when there are name conflicts.

- The user resolves ambiguities in expressions caused by missing operation precedence by explicitly parenthesizing the text.

## Extra Features

### 1. G++ File Reading:

- G++ files can be read and executed by the interpreter.
- G++ scripts can be loaded by users for batch processing.



The screenshot shows a window titled 'input.g++'. The left pane contains a G++ script with the following code:

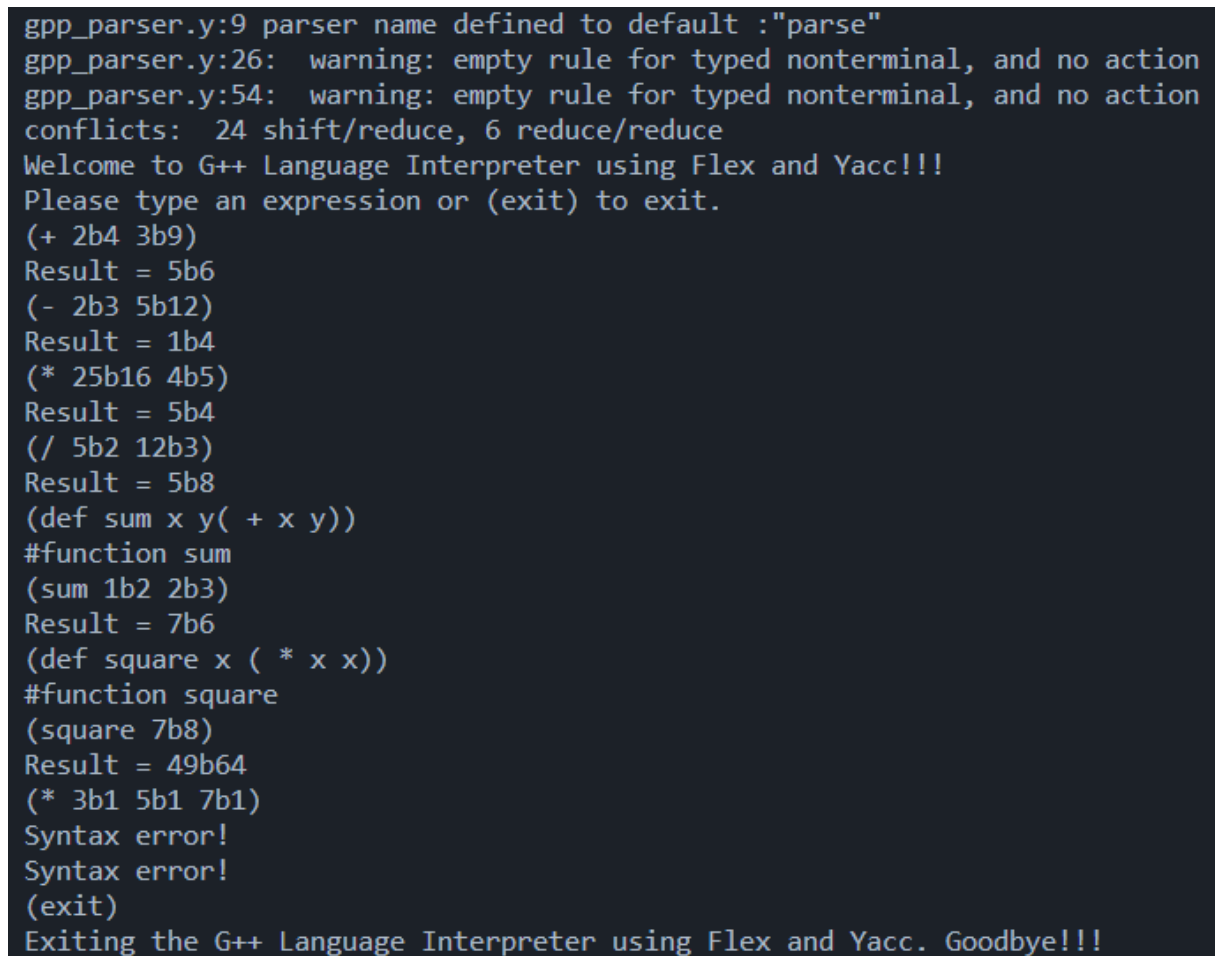
```
1 (+ 4b1 6b1)
2 (def sum x y (+ x y))
3 (sum 4b1 6b1)
4 (* 4b1 6b1)
5 (def square x (* x x))
6 (square 5b1)
7 (* 4b1 6b1 8b1)
8 (exit)
```

The right pane shows the interpreter's output:

```
gpp_parser.y:9 parser name defined to default : "parse"
gpp_parser.y:26: warning: empty rule for typed nonterminal, and no action
gpp_parser.y:54: warning: empty rule for typed nonterminal, and no action
conflicts: 24 shift/reduce, 6 reduce/reduce
Welcome to G++ Language Interpreter using Flex and Yacc!!!
Please type an expression or (exit) to exit.
Result = 10b1
#function sum
Result = 10b1
Result = 24b1
#function square
Result = 25b1
Syntax error!
Syntax error!
Exiting the G++ Language Interpreter using Flex and Yacc. Goodbye!!!
```

### 2. Terminal Input:

- Direct command execution from the terminal is also supported by the interpreter.
- This feature makes it possible to test and execute G++ language commands interactively.



The screenshot shows a terminal window with the following interactive session:

```
gpp_parser.y:9 parser name defined to default : "parse"
gpp_parser.y:26: warning: empty rule for typed nonterminal, and no action
gpp_parser.y:54: warning: empty rule for typed nonterminal, and no action
conflicts: 24 shift/reduce, 6 reduce/reduce
Welcome to G++ Language Interpreter using Flex and Yacc!!!
Please type an expression or (exit) to exit.
(+ 2b4 3b9)
Result = 5b6
(- 2b3 5b12)
Result = 1b4
(* 25b16 4b5)
Result = 5b4
(/ 5b2 12b3)
Result = 5b8
(def sum x y (+ x y))
#function sum
(sum 1b2 2b3)
Result = 7b6
(def square x (* x x))
#function square
(square 7b8)
Result = 49b64
(* 3b1 5b1 7b1)
Syntax error!
Syntax error!
(exit)
Exiting the G++ Language Interpreter using Flex and Yacc. Goodbye!!!
```

## **Lisp Part**

**Overview:** This document describes the Lisp-developed G++ Language Interpreter. This interpreter is intended to support fractional-order mathematical operations and to make function definition and execution in the G++ language format easier.

### **Linguistic Features**

#### **1. Types and Structures of Data:**

- Fractions: Shown as a numerator and denominator, two integers.
- Identifiers: Variable and function names are represented by symbols.

#### **2. Operations:**

- Arithmetic Operations: Manages addition, subtraction, multiplication, and division operations involving fractions.
- Fraction Simplification: During operations, fractions are automatically simplified.

#### **3. Definition and Use of Functions:**

- Function Definition: Lisp syntax can be used to define functions.
- Carrying Out Tasks: Calling a function's name and passing arguments causes it to run.

### **Expression Assessment Method**

#### **1. Recursive Evaluation:**

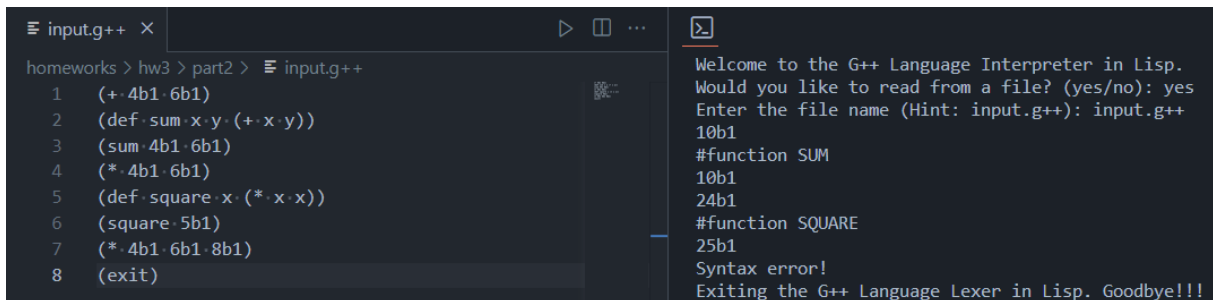
- Expressions are evaluated in a recursive manner, effectively managing nested structures.
- Function calls are assessed by running the function body after the arguments have been resolved.

#### **2. Error Handling:**

- Mistakes are addressed and communicated to the user, such as division by zero.

## Reading Files and Using Interactive Mode

1. File Reading: Batch processing and script execution are made possible by the interpreter's ability to read and run G++ scripts from files.



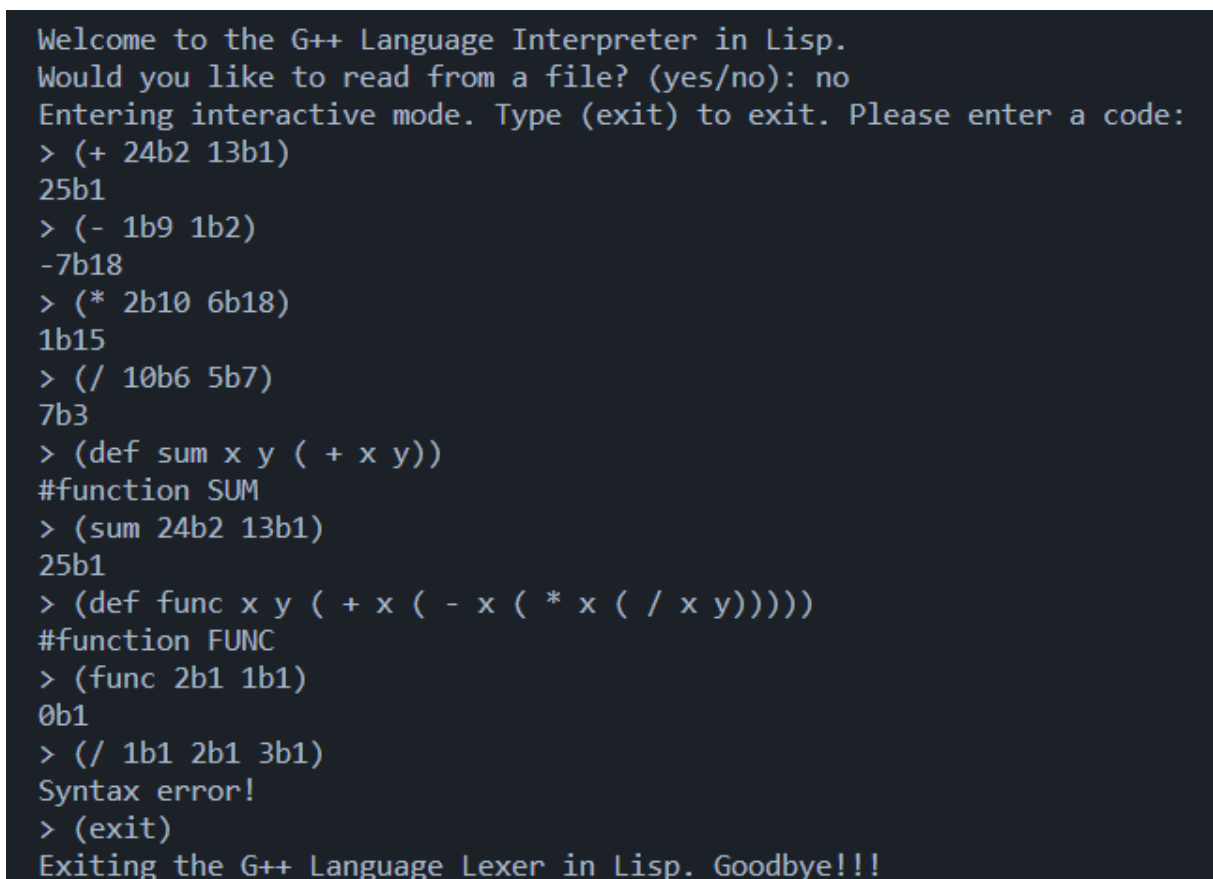
The screenshot shows a terminal window with the G++ Language Interpreter in Lisp. The left pane displays the contents of a file named 'input.g++', which contains several lines of code including arithmetic operations and function definitions. The right pane shows the interpreter's output, which includes a welcome message, a prompt to read from a file (answered 'yes'), the file name 'input.g++', and the execution results of the code in the file. The output shows the results of arithmetic operations and the definition of functions 'SUM' and 'SQUARE'. A syntax error is also shown for a line in the file.

```
input.g++ x
homeworks > hw3 > part2 > input.g++
1  (+ 4b1 6b1)
2  (def sum x y (+ x y))
3  (sum 4b1 6b1)
4  (* 4b1 6b1)
5  (def square x (* x x))
6  (square 5b1)
7  (* 4b1 6b1 8b1)
8  (exit)

Welcome to the G++ Language Interpreter in Lisp.
Would you like to read from a file? (yes/no): yes
Enter the file name (Hint: input.g++): input.g++
10b1
#function SUM
10b1
24b1
#function SQUARE
25b1
Syntax error!
Exiting the G++ Language Lexer in Lisp. Goodbye!!!
```

## 2. Interactive Terminal Mode:

- Commands can be executed interactively from the terminal by the interpreter.
- Expressions can be directly entered by users for instant evaluation.



The screenshot shows the G++ Language Interpreter in Lisp in interactive terminal mode. The output displays a welcome message, a prompt to read from a file (answered 'no'), and a message indicating that interactive mode is entered. The user then enters several commands, including arithmetic operations and function definitions, which are evaluated and the results are displayed. The output shows the results of arithmetic operations and the definition of functions 'SUM' and 'FUNC'. A syntax error is also shown for a line in the input.

```
Welcome to the G++ Language Interpreter in Lisp.
Would you like to read from a file? (yes/no): no
Entering interactive mode. Type (exit) to exit. Please enter a code:
> (+ 24b2 13b1)
25b1
> (- 1b9 1b2)
-7b18
> (* 2b10 6b18)
1b15
> (/ 10b6 5b7)
7b3
> (def sum x y (+ x y))
#function SUM
> (sum 24b2 13b1)
25b1
> (def func x y (+ x (- x (* x (/ x y)))))
#function FUNC
> (func 2b1 1b1)
0b1
> (/ 1b1 2b1 3b1)
Syntax error!
> (exit)
Exiting the G++ Language Lexer in Lisp. Goodbye!!!
```

## Functionality Specifics

### 1. Parsing and Evaluating Expressions:

- Expressions undergo dynamic parsing and evaluation.
- A hash table is used to store function definitions so they can be quickly retrieved and executed.

### 2. Fraction Operations:

- Fractions are automatically simplified and parsed from strings.
- Certain Lisp functions are used to implement arithmetic operations on fractions.

200104004085

Mert Emir Şeker