G++ Language Syntax Analyzer Documentation

Muhammed Bilal Türk - 210104004047

1. Project Overview

1.1 Purpose

This project implements a syntax analyzer for the G++ programming language, designed for educational purposes at Gebze Technical University. The goal is to create an interpreter that can parse and analyze G++ code according to its specific syntax rules.

1.2 Language Characteristics

G++ is characterized by the following features:

- Lisp-like syntax
- Interpreted language
- Imperative and non-object-oriented
- Static scope and static binding
- Strongly typed
- Built-in support for precise arithmetic

2. Implementation Details

2.1 Tools and Technologies

- Flex (lexical analyzer generator)
- Yacc (parser generator)
- C programming language

2.2 Lexical Analysis

The lexical analyzer (lexer) recognizes the following elements:

<u>Keywords</u>

- Logical: `and`, `or`, `not`, `equal`, `less`

- Control flow: `if`, `for`, `exit`
- Data structures: `list`, `append`, `concat`
- Others: `nil`, `set`, `deffun`, `load`, `print`, `true`, `false`

Operators

- Arithmetic: `+`, `-`, `/`, `*`
- Grouping: `(`, `)`, `,`

Literals

- Unsigned integers
- Unsigned fractions (represented as `numerator:denominator`)

Identifiers

- Alphanumeric strings starting with an alphabetic character

2.3 Syntax Analysis

The syntax analyzer is designed to handle:

- Arithmetic expressions
- Logical operations
- Conditional statements
- Function definitions and calls
- Basic data type manipulations

2.4 Fraction Handling

A custom `Fraction` struct is used to represent and manipulate fractions:

```
typedef struct {
  int numerator;
  int denominator;
```

} Fraction;

3. Key Implementation Features

3.1 Expression Evaluation

- All expressions return a fraction
- Supports basic arithmetic: addition, subtraction, multiplication, division
- Implements logical operations: `and`, `or`, `not`
- Conditional expressions using `if`

3.2 Error Handling

- Syntax errors are reported with descriptive messages
- Division by zero is explicitly handled
- Invalid characters are detected and reported

3.3 Interpreter Mode

Two operational modes:

1. Interactive mode: Prompts user for input
2. File input mode: Reads and interprets code from a specified file

4. Limitations and Assumptions

4.1 Unspecified Language Features

Some language features were not explicitly defined in the specification and were resolved through implementation:

- Expression evaluation strategy
- Scoping rules
- Variable binding mechanisms

4.2 Data Type Constraints

- Only unsigned fractions and integers are supported
- Strong typing is enforced

5. Conclusion

The G++ syntax analyzer successfully implements a basic interpreter for the specified language, demonstrating lexical and syntactic parsing capabilities while maintaining the language's educational design principles.