Info-F403 – Introduction to language theory and compiling

Genial Imperative Language for Learning and the Enlightenment of Students Part-I Report

Herma ELEZI

Gilles GEERAERTS
Mrudula BALACHANDER
Mathieu SASSOLAS

2024

Contents

Abstract

This document provides a comprehensive overview of the implementation of the lexical analyzer for the GILLES programming language, designed for the Info-F403 course. The analyzer was developed using JFlex and handles token recognition, symbol table management, error detection, and test cases for the specified language syntax.

1 Introduction

This project aims to create a lexical analyzer for GILLES using JFlex. The analyzer should recognize lexical units such as program names, variables, numbers, and comments, while also handling errors and maintaining a symbol table. This report outlines the regular expressions used, structure of the analyzer, handling of comments, error handling, and example test files.

2 Regular Expressions

Extended regular expressions (Regex) were used to define the necessary lexical units for GILLES.

- **ProgName**: `[A-Z][A-Za-z]*`
- **VarName**: `[a-z][A-Za-z0-9]*`
- **Number**: `[0-9]+`
- **WhiteSpace**: `(" " | "\t" | "\r" | "\n")+`

The order of these expressions is crucial; the first matching expression determines the token type.

2.1 ProgName

A program name starts with an uppercase letter, followed by any combination of letters and underscores. This ensures that program names are easily identifiable, starting with an uppercase letter and potentially containing underscores or additional letters.

2.2 VarName

Variable names begin with a lowercase letter and may be followed by a combination of letters and digits. This regex allows any lowercase letter as the first character, ensuring variable names are recognized appropriately.

2.3 Number

Numbers are represented as sequences of one or more digits using the regex `[0-9]+`, making it possible to recognize integral constants.

2.4 WhiteSpace

To focus on meaningful tokens, whitespace characters (spaces, tabs, carriage returns, and newlines) are combined in a single regex. This ensures that multiple whitespace characters are treated as a single unit, simplifying token recognition.

3 Implementation of the Lexical Analyzer

The lexical analyzer was implemented in a JFlex file, `LexicalAnalyzer.flex`. It recognizes various tokens, including program names, variable names, numbers, keywords (e.g., `LET`, `IF`), and operators. The file is divided into three main sections: configuration, lexical rules, and token definitions.

3.1 Configuration

Key JFlex options are configured to manage token behavior:

- `%type Symbol`: Specifies that tokens are of type `Symbol`, which includes lexical type, value, line, and column.
- `%function nextToken`: Defines the `nextToken` function, which retrieves tokens during analysis.
- `%eofval`: Specifies the return value for end-of-file, ensuring that an End of Stream (EOS) token is returned.

3.2 Lexical States

Different lexical states handle various aspects of GILLES syntax, especially for comments:

- `YYINITIAL`: Default state for recognizing keywords, variable names, numbers, and operators.
- `SHORTCOMMENTS` and `LONGCOMMENTS`: These states recognize comments. Short comments end at the newline, while long comments are enclosed in `!!` and span multiple lines if needed.

4 Comment Handling

GILLES supports two types of comments: short comments (starting with `$`) and long comments (enclosed in `!!`). The content of both comment types is ignored.

4.1 Short Comments

Short comments start with `$` and end at the newline. The lexer reverts to `YYINITIAL` upon encountering a newline, ignoring the comment text.

4.2 Long Comments

Long comments can span multiple lines, remaining in the `LONGCOMMENTS` state until `!!` is encountered. If the end of the file is reached without a closing `!!`, an exception is raised.

5 Error Handling

Unrecognized characters are flagged with a `PatternSyntaxException`. Two error cases are handled:

- An unknown character results in a `PatternSyntaxException`.
- If a long comment isn't closed by the end of the file, an exception is raised, indicating an incomplete comment.

6 Output

In `main.java`, the source file is read and tokenized by calling `lexer.nextToken()`. Each token is printed until the `EOS` (End of Stream) token is reached. Variables are stored in a symbol table and displayed with their first occurrence line number.

7 Test Files

Several test files were created to verify the lexer's accuracy:

- `AdditionalTokens.gls`: Tests recognition of redundant tokens.
- `InvalidNumber.gls`: Verifies detection of invalid numbers.
- `InvalidProgName.gls` and `InvalidVarName.gls`: Test invalid program and variable names.
- `LongComments.gls` and `ShortComments.gls`: Test comment handling.
- `UnclosedLongComments.gls`: Verifies detection of unclosed long comments.
- `ValidProgName.gls` and `ValidVarName.gls`: Ensure valid program and variable names are correctly handled.
- `RandomWhitespaces.gls`: Tests proper handling of whitespace.

8 Nested Comments

Handling nested comments requires a balance mechanism for opening and closing symbols. Although GILLES does not currently support nested comments, a counter mechanism could be implemented in future versions to track the depth of comment nesting.