INFO-F403: Introduction to language theory and compiling
Part 1: lexical analyzer of GILLES

Loïc Blommaert (542721, M-INFOS), Hugo Charels (544051, M-INFOS)

October 24, 2024

1 Introduction

This report covers the design and implementation of the lexical analyzer for *GILLES* using *JFlex*. The following sections will elaborate on language specification, recognizable regular expressions (regex) and briefly explain the implementation choices.

2 Language Specification

GILLES must recognize the following keywords :

```
LET, BE, END, IF, THEN, ELSE, WHILE, REPEAT, OUT, IN
```
(2.0.1)

The same goes for the following operators :

```
:, =, (, ), -, +, *, /, {, }, ->, |, ==, <=, <
```
(2.0.2)

While comments are indicate by `$<comment>`, for short comments and by `!!<comment>!!` for long comments (where `<comment>` could be everything). Note that nested (long) comment are not supported by this language for technical reasons, explained in section 2.3.

Variables and program names follow specific naming conventions, which are further defined using regex in the lexer.

2.1 Token recognition

The lexer processes the input by reading symbols sequentially. Using the automaton generated by *JFlex*, each recognized token is matched with its corresponding lexical unit, which can be one of the following :

- `PROGNAME, VARNAME`, respectively for the name of the program, a name of a variable.
- `NUMBER`, for a number (value).
- `LET, BE, END, IF, THEN, ELSE, WHILE, REPAT, OUTPUT, INPUT`, for the keywords (corresponding respectively to 2.0.1).
- `COLUMN, ASSIGN, LPAREN, RPAREN, MINUS, PLUS, TIMES, DIVIDE, LBRACK, RBRACK, IMPLIES, PIPE, EQUAL, SMALEQ, SMALLER`, for the operators (corresponding respectively to 2.0.2).
- `EOS`, for the end of stream.

As long as the new token is recognized and is not an `EOS`, the lexer displays the read token and its corresponding lexical unit in the terminal. On reading, if a new variable is encountered, the variable (and therefore its name) is stored in a map, with the line of its instantiation.
Note that comments are of course not displayed because the scanner ignore them. Same for the set of whitespace: {␣, \t , \r, \n}.

The lexer may halts and throws an error if an unrecognizable symbol is part of the input file or if a long comment is never closed.

2.2 Regex

The following regular expressions describe the language:

Reference	Regex	Meaning
`AlphaUpperCase`	`[A-Z]`	A capital letter of the roman alphabet
`AlphaLowerCase`	`[a-z]`	A lowercase letter of the roman alphabet
`Alpha`	`AlphaUpperCase|AlphaLowerCase`	A letter of the roman alphabet
`Numeric`	`[0-9]`	A western digit
`AlphaNumeric`	`Alpha|Numeric`	A letter or a digit
`ProgName`	`AlphaUpperCase(Alpha|_)*`	A capital letter, followed (or not) by a sequence of letter or _
`VarName`	`AlphaLowerCase(AlphaNumeric)*`	A lowercase letter, followed (or not) by a sequence of `AlphaNumeric`
`Number`	`Numeric+`	At least one digit
`Whitespace`	`[␣\t \r \n]+`	At least one element of the set {␣, \t , \r, \n}

Table 1: Regex of *GILLES*

2.3 Nested Comments

Nested comments (long comments in long comments) are not supported by the language due to their difficulty. Nested comments are similar to the parenthesizing problem. We need to keep track of the actual depth of the comment with a stack-based approach, and so, the lexer could not be implemented with a simple automaton. In addition, it requires to indicate the beginning and the end of the comment differently, otherwise it would be impossible to keep track of the depth.

3 Example of lexer analyzes

The figure 1 presents a *GILLES* file on which the lexer displays the result shown on the figure 2. As we can see, the file contains various comments, keywords and operators. It also includes a program name (Evaluation_algorithm) and a variable name (grade). The lexer recognizes the tokens and determines the corresponding lexical units.

```
!! Evaluation algorithm !!

LET Evaluation_algorithm BE
  IN(grade):
  WHILE {grade < 20} REPEAT    $ It should not be the case
    grade = grade + 1:
  END
  IF {grade == 20} THEN
      grade = 21:              $ It's really impressive, it deserves a bonus
  END:
  OUT(grade):
END
```

Figure 1: Example of *GILLES* file

For the curious, some tests have been written to show the result of scripts playing with "edge cases". The expected results are provided in *.out* files, which are described by our previous explanations.

4 Conclusion

In this report, we defined tokens, lexical units and regex specifications. We discussed the limitations of handling nested comments and explained briefly the functioning of the lexer.

In future work, enhancements could be made to the language by introducing error recovery mechanisms or new lexical unit. Overall, the current implementation demonstrates a functional lexical analysis process for *GILLES*, laying a solid foundation for future development.

```
token: LET               lexical unit: LET
token: Evaluation_algorithm     lexical unit: PROGNAME
token: BE                lexical unit: BE
token: IN                lexical unit: INPUT
token: (                 lexical unit: LPAREN
token: grade             lexical unit: VARNAME
token: )                 lexical unit: RPAREN
token: :                 lexical unit: COLUMN
token: WHILE             lexical unit: WHILE
token: {                 lexical unit: LBRACK
token: grade             lexical unit: VARNAME
token: <                 lexical unit: SMALLER
token: 20                lexical unit: NUMBER
token: }                 lexical unit: RBRACK
token: REPEAT            lexical unit: REPEAT
token: grade             lexical unit: VARNAME
token: =                 lexical unit: ASSIGN
token: grade             lexical unit: VARNAME
token: +                 lexical unit: PLUS
token: 1                 lexical unit: NUMBER
token: :                 lexical unit: COLUMN
token: END               lexical unit: END
token: IF                lexical unit: IF
token: {                 lexical unit: LBRACK
token: grade             lexical unit: VARNAME
token: ==                lexical unit: EQUAL
token: 20                lexical unit: NUMBER
token: }                 lexical unit: RBRACK
token: THEN              lexical unit: THEN
token: grade             lexical unit: VARNAME
token: =                 lexical unit: ASSIGN
token: 21                lexical unit: NUMBER
token: :                 lexical unit: COLUMN
token: END               lexical unit: END
token: :                 lexical unit: COLUMN
token: OUT               lexical unit: OUTPUT
token: (                 lexical unit: LPAREN
token: grade             lexical unit: VARNAME
token: )                 lexical unit: RPAREN
token: :                 lexical unit: COLUMN
token: END               lexical unit: END

Variables
grade   4
```

Figure 2: Result of the lexer analysis