LW1

DA53: Compilers and Language Theory

stephane.galland@utbm.fr

# Lexical Analysis by hand

# 00 | Goal

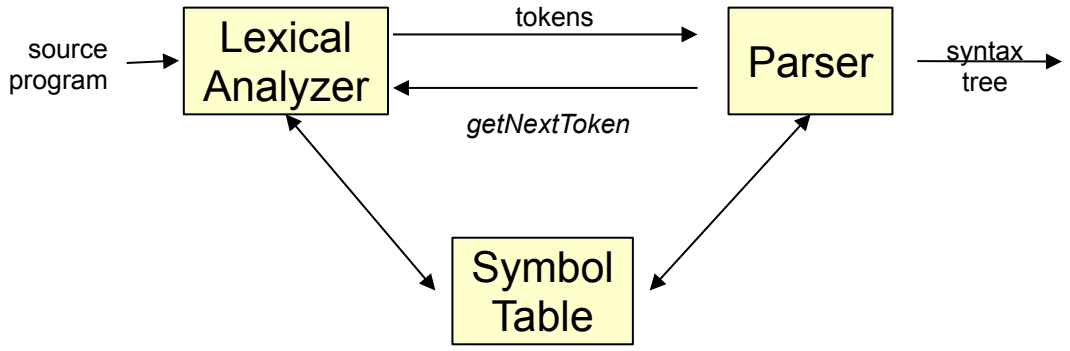
The goal of this on-computer tutorial session is to write a small lexical analyzer (or lexer) by hand.

The development could be done in Java, C# or Python.

Remember that the lexical analyzer takes a source program, and scans it, and replies the recognized tokens when they are requested by the parser.

Definitions :

- **Lexeme :** Sequence of characters that is recognized by the parser (usually expressed by regular expression)
- **Token :** Symbolic name of a lexeme




During this on-computer tutorial, we must :

- Define the language of the source program, its syntax and its semantic.
- Write a symbol table
- Write the lexer

# 01 | Language Description

The language to implement in this Labwork (LW) is a part of the Tiny Basic dialect of the BASIC language.

## 01.1 Statements

The language is composed of statements, one per line. If this ST we do not recognize multiple statements on the same line. A line must follow one of the following formats :

- statement
- number statement

where number is the number of the line, and statement is the instruction of the language.

The supported statements are :

- **PRINT** expression
  - Print the given expression on the standard output
- **IF** expression relational_operator expression **THEN** statement
  - The statement is executed only if the given relational operator on the two given expressions replies true.
  - The relational operators are :
    - = equality
    - < less than
    - > greater than
    - <= less or equal
    - >= greater or equal
    - <>, >< not equal
- **GOTO** expression
  - Run the statement at the line defined by the expression.
- **INPUT**variable [**,** variable ...]
  - Read variable values from the standard input .
- **LET** variable **=** expression
  - Assign the value of the expression to the variable.
- **GOSUB** expression
  - Run the statement at the line defined by the expression, but in opposite than GOTO, remember the line of the GOSUB and run the statement just after that line when the statement RETURN is encountered.
- **RETURN**
  - Run the statement at the line following the last GOSUB.
- **END**
  - Stop the program.
- **REM** text
  - Insert a comment in the source code.

## 01.2 Expressions

Many of these statement types involve the use of expressions. An expression is the combination of one or more *numbers*, *strings* or *variables*, joined by *operators*, and possibly grouped by *parentheses*.

There are four operators: +, -, *, /

Note that the operators + and – have two versions : the binary (a+b) and the unary (+a).

A string of characters is enclosed by the quote characters (").

## 01.3 Variable Definition and Scope

The variables do not need to be declared. The scope of all the variables is global. So that a variable, when it is used, *could be undefined*.

All the variables are of numerical type or of string type, depending on the type of the expression assigned to the variable.

## 01.4 Example

Let the following Tiny Basic program :

```
5 LET S = 0
10 INPUT NUM
15 INPUT V
20 LET N = NUM
30 IF N <= 0 THEN GOTO 99
40 LET S = S + V
50 LET N = N - 1
70 GOTO 30
99 PRINT S/NUM
100 END
```

Your lexer must reply the following tokens, in that order :

```
<NUM,5> <LET> <ID,S> <=> <NUM,0>
<NUM,10> <INPUT> <ID,NUM>
<NUM,15> <INPUT> <ID,V>
<NUM,20> <LET> <ID,N> <=> <ID,NUM>
<NUM,30> <IF> <ID,N> <RELOP,<=> <NUM,0> <THEN> <GOTO> <NUM,99>
<NUM,40> <LET> <ID,S> <=> <ID,S> <OP,+> <ID,V>
<NUM,50> <LET> <ID,N> <=> <ID,N> <OP,-> <NUM,1>
<NUM,70> <GOTO> <NUM,30>
<NUM,99> <PRINT> <ID,S> <OP,/> <ID,NUM>
<NUM,100> <END>
```

# 02 | Exercices

## 02.1 Exercice 1

- What is the alphabet of the language ?
- What are the lexemes (as regular expressions) of the language ?
- Write the table that is matching the lexemes, the tokens, and the attributes of the tokens.

## 02.2 Symbol Table

- Create the class SymbolTable that permits to store several informations about the variables : name, lexeme, line of the first occurrence, etc.
- This symbol table is a kind of map with the name of the variable as key and the description of the variable as value.

## 02.2 Tokens

- Define the hierarchy of classes that is describing all the tokens of the language.
- Note that the token ID should point to an entry of the symbol table.

## 02.3 Scanner

- Create the Scanner class that takes an input stream (Reader is prefered in Java) as parameter of its constructor.
- The scanner is able to read the characters from the stream, to compact white spaces when possible.
- The scanner is able to reply the line in the source program at which the read character is located.
- We recommend to provides to functions : char peek(), and char get(). The first replies the next available character but does not consume it. The second replies the next available character and consumes it to pass to the next available character.

## 02.4 Lexer

- Create the Lexer class and a function inside that is able to read a character from an input stream.
- Create a Finite Automata (FA) or an ad-hoc (from-scratch) code that is recognizing the language. If you are encountering problems to do the FA, you may take the FA at the end of this document.
- Implements the FA or the ad-hoc in the function getNextSymbol() of your Lexer, and it should use the Scanner.
- Remember that the Lexer should not reply a token for the comment statements.

## 02.5 Main Program

- Write the main program that creates the Symbol Table, and the Lexer on an input file.
- The main program invokes the getNextSymbol() function of the Lexer to retreive the tokens.
- The tokens are displayed on the standard output.

# 03 | Finite Automata

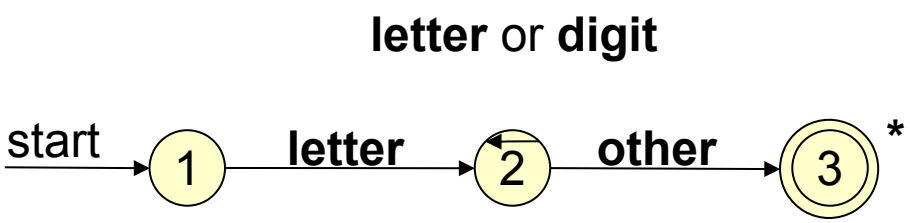
A Finite Automata (explained in Chapter 2) is able to recognized one or more tokens. When a state is drawn with two circles, it means that a token is recognized and the Lexer could return it. If a star is drawn on the side of a two-circle state, it means that it does not consume the character from the input stream that has permits to reach this star-two-circle state.

## 03.1 Alphabet

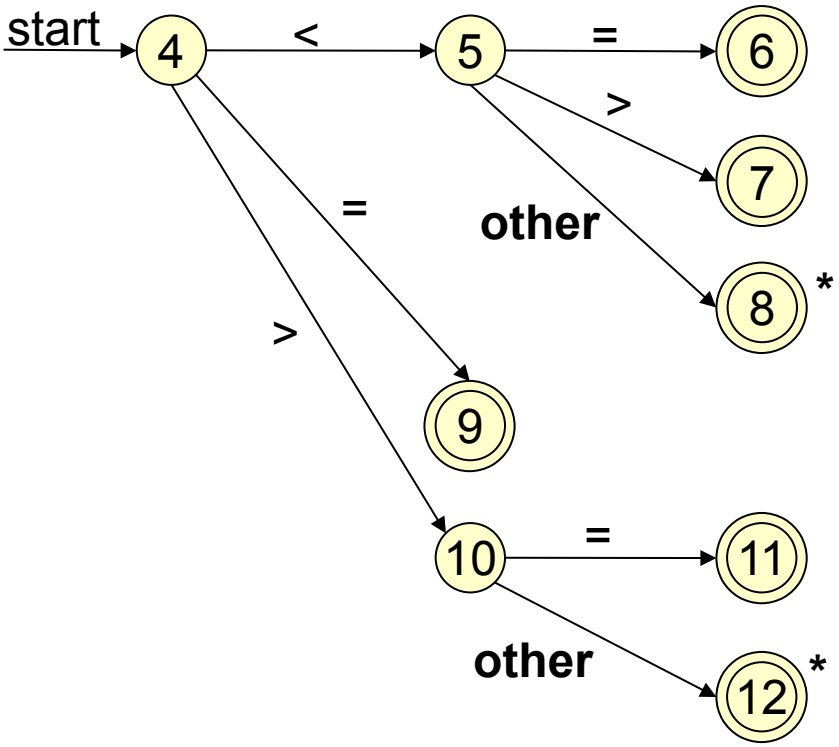
The alphabet of the language is :

- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- +, -, /, *
- a, b, c, d, …., x, y, z
- A, B, C, D, ….., X, Y, Z
- ., <, >, =, (, ), "

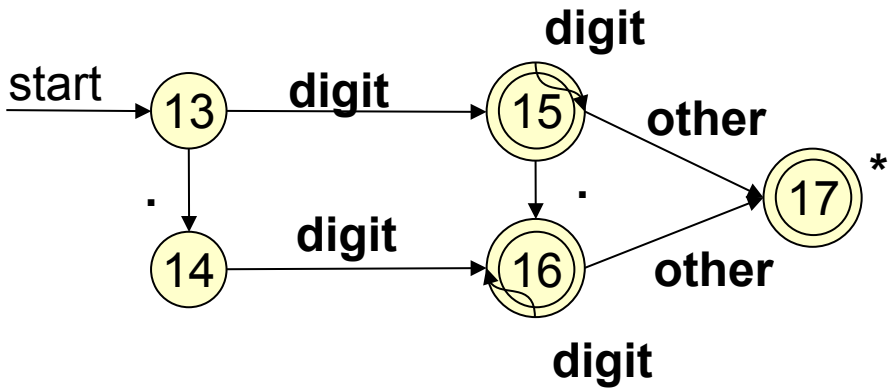
## 03.2 Finite Automata for Token ID



03.3 Finite Automata for Token RELOP




03.4 Finite Automata for Token NUM



03.5 Finite Automata for Tokens OP and =